Subsumption Algorithms for Some Attributive Concept Description Languages

Bernhard Hollunder

SEKI Report SR-89-16

Subsumption Algorithms for Some Attributive Concept Description Languages

Bernhard Hollunder

Deutsches Forschungszentrum

für künstliche Intelligenz

Abstract.

This paper investigates subsumption algorithms for logic-based knowledge representation languages of the KL-ONE family. We amalgamate the attributive concept description language $\mathcal{ALC}$, that contains value restrictions, intersections, unions and complements with number restrictions, role hierarchies (to model the KL-ONE's roleset differentiation), and Feature Logic, respectively. We show that deciding consistency and subsumption of $\mathcal{ALC}$ extended with number restrictions and $\mathcal{ALC}$ extended with role hierarchies is PSPACE-complete. Furthermore, for all these languages we give subsumption algorithms.

Acknowledgements.

I am very grateful to Manfred Schmidt-Schauß and Werner Nutt. Manfred introduced me to the area of knowledge representation languages. Furthermore, he gave me hopeful ideas to do this work. With Werner I had fruitful discussions about the contents and the structure of this paper. His ideas influenced this paper, too. I would like to thank both, Manfred and Werner, for reading earlier drafts of this thesis.

Contents

1 Introduction

Main research topics in knowledge representation are the development of formalisms to describe knowledge and to reason with the represented knowledge. After the definition of a knowledge representation language and its semantics, one can describe a specific domain by filling the knowledge base with explicit knowledge. The derivation of knowledge only represented implicitly will be called reasoning. Basically, it can be more difficult to reason correctly with one knowledge representation language than with another. Moreover, this difficulty increases as the expressive power of the language. This amounts to a tradeoff between the expressiveness of a knowledge representation language and its computational complexity.

In this paper we are interested in logic-based knowledge representation languages based on KL-ONE [Levesque/Brachman 87, Nebel 88, Donini/Lenzerini 88, Schmidt-Schauß/Smolka 88]. These languages aim at describing sets of objects by specifying restrictions on attributes the objects may have. We will call this formalism attributive concept description (ACD, for short). ACD-languages employ two kinds of symbols, called concepts and roles. Concepts are interpreted as sets and roles are interpreted as binary relations. To form concept descriptions we allow a few operators. The concept description defined by

- $\forall$R:C can be read as the set of "all objects for which all R's are in C",
- $\exists$R:C can be read as the set of "all objects for which there is an R in C",
- (atleast N R) can be read as the set of
  "all objects for which there are at least N R's", and
- (atmost N R) can be read as the set of
  "all objects for which there are at most N R's",

where R is a role, C a concept description, and N is non-negative integer.

Now, let us consider a small example. The class of vehicles is partitioned into the disjoint subclasses of vehicles with motor and vehicles without motor. A motor-cycle is a vehicle with a motor and with exactly two wheels. A motor-car is a vehicle with a motor and with at least 3 and at most 4 wheels. A truck is a vehicle with a motor and with at least 6 wheels. This can be expressed by the following axioms:

vehicle_without_motor $\sqsubseteq$ vehicle

vehicle_with_motor = vehicle $\sqcap$ $\neg$ vehicle_without_motor

motor-cycle = vehicle_with_motor $\sqcap$ (atleast 2 wheels) $\sqcap$ (atmost 2 wheels)

motor-car = vehicle_with_motor $\sqcap$ (atleast 3 wheels) $\sqcap$ (atmost 4 wheels)

truck = vehicle_with_motor $\sqcap$ (atleast 6 wheels),

where the $\sqcap$ is the intersection and the $\neg$ is the complement of sets.

Given an interpretation I for the occurring concept and role symbols, these concept descriptions are interpreted as the sets

- $I[\forall R: C] = \{ a \in \mathcal{D}^I \mid \forall (a,b) \in I[R] : b \in I[C] \}$
- $I[\exists R: C] = \{ a \in \mathcal{D}^I \mid \exists (a,b) \in I[R] : b \in I[C] \}$
- $I[\text{atleast N R}] = \{ a \in \mathcal{D}^I \mid \| \{ b \in \mathcal{D}^I \mid (a,b) \in I[R] \} \| \geq N \}$
- $I[\text{atmost N R}] = \{ a \in \mathcal{D}^I \mid \| \{ b \in \mathcal{D}^I \mid (a,b) \in I[R] \} \| \leq N \}$,

where $\mathcal{D}^I$ is the domain of I. Since concept descriptions are interpreted as sets, it is obvious how to handle concept intersection, concept union and concept complement. We will speak of an ACD-language if at least $\forall R:C$, $\exists R:\top$ (where $I[\top] = \mathcal{D}^I$ in every interpretation I) and concept intersection are available.

So far we have established how to form concept descriptions in ACD-languages. Now we will see how to do reasoning in these languages. Given two concept descriptions C and D, we want to know whether C is subsumed by D, that is, if every interpretation interprets C as a subset of D. In KL-ONE systems the subsumption test is needed to find out subsumption relations between several concept descriptions. Besides the subsumption

test we are interested in two further problems:

- Is a concept description C consistent, that is, does there exist an interpretation I such that $I[C] \neq \emptyset$?
- Is a concept description C equivalent to a concept description D, that is, is $I[C] = I[D]$ for every interpretation I ?

If an ACD-language contains complement, then the different problems can be reduced to each other in linear time.

We go back to our starting-point and consider the tradeoff between the expressiveness of ACD-languages and their computational tractability. The expressiveness is determined by the operators we admit to the language, and the tractability is determined by the computational complexity of the subsumption algorithm, that is, what time is needed to get an answer to the question: is concept description C subsumed by concept description D ?

In [Levesque, Brachman 87] the minimal ACD-language $\mathcal{FL}^-$, which contains the operators $\forall R:C$, $\exists R:\top$ and concept intersection, is examined, and it is shown that subsumption can be decided in quadratic time. In [Schmidt-Schauß/Smolka 88] further ACD-languages with enhanced expressiveness as compared to $\mathcal{FL}^-$ are investigated. For example, the language $\mathcal{ALC}$ that admits the operators $\forall R:C$, $\exists R:C$, intersection, union and complement is defined, and it is shown that checking consistency of $\mathcal{ALC}$ - concept descriptions is PSPACE-complete.

Very often it is demanded that the subsumption test is tractable, that is, that subsumption can be checked in polynomial time. This means that the expressiveness of ACD-languages must be restricted dramatically. For example, ACD-languages satisfying this demand are less powerful than propositional logic. In particular, these languages must not contain simultaneously intersections, unions and complements like the propositional

logic. One remedy is the use of incomplete subsumption algorithms. The practical usefulness of such incomplete algorithms depends on what kind of subsumption relations the algorithms don't find out. It is of great help to examine complete subsumption algorithms and then to restrict them, such that the algorithms are efficient. Then, of course, one can investigate what kind of subsumption relations the algorithms don't find out.

In this thesis we examine the computational complexity of consistency checking algorithms for some ACD-languages. Starting from the language $\mathcal{ALC}$, we amalgamate this language with some well-known operators occurring in KL-ONE.

In chapter 2 we define the syntax and semantics of the language $\mathcal{ALCN}$ ($\mathcal{ALC}$ + $\mathcal{N}$umber restriction), which is the language $\mathcal{ALC}$ amalgameted with *number restrictions* .

Since the applicative structure of concept descriptions is not very suitable for devising consistency checking algorithms, the concept descriptions will be translated into constraint systems. The definition of constraint systems and the translation of concept descriptions into constraint systems is performed in chapter 3.

Chapter 4 shows that checking consistency of $\mathcal{ALCN}$ - concept descriptions is decidable.

In chapter 5 we characterize the problem of checking consistency of $\mathcal{ALCN}$- concept descriptions with respect to the computational complexity and we show that this problem is PSPACE-complete.

In chapter 6 we define *role hierarchies* , which are partial orders on role symbols. If $P \leq R$ is in a role hierarchie, then P denotes a subrelation of the relation denoted by R. The idea behind this is to model the *roleset - differentiation* in KL-ONE. We prove that checking consistency of $\mathcal{ALCR}$ - concept descriptions ($\mathcal{ALC}$ + $\mathcal{R}$ole hierarchies) is PSPACE-complete. Furthermore we give an algorithm for checking consistency of $\mathcal{ALCNR}$- concept descriptions ($\mathcal{ALC}$+ $\mathcal{N}$umber restrictions + $\mathcal{R}$ole hierarchies).

Feature Logic as described in [Smolka 88] is very similar to the language $\mathcal{ALC}$. In Feature Logic it is assumed that the roles are functional. That is, the interpretation of a functional role, which is called a *feature*, is not only a relation, but a partial function. Furthermore the logic contains the *agreement* and *disagreement* operators, counterparts to role value maps in the KL-ONE world. Smolka shows that Feature Logic has an NP-complete consistency problem and a co-NP-complete subsumption problem. In chapter 7 we examine the language $\mathcal{ALCF}$, that is a combination of $\mathcal{ALC}$ and $\mathcal{F}$eature Logic. We show that checking consistency of $\mathcal{ALCF}$- concept descriptions is decidable.

2 Attributive concept descriptions

In the first part of this chapter we define the syntax and semantics of the attributive concept description language $\mathcal{ALCN}$, that admits ∀R:C, ∃R:C, intersections, unions, complements and number restrictions. After that we compare this language with some related ACD-languages.

2.1. The language $\mathcal{ALCN}$

Let two disjoint alphabets of symbols, called **concepts** and **roles**, respectively, be given. The special concept symbols ⊤ and ⊥ are called top symbol and bottom symbol. The letters A und B will always denote concept symbols, the letter R will always denote a role symbol, and the letter N will always denote a non-negative integer represented in binary.

The concept descriptions of the ACD-language $\mathcal{ALCN}$ are given by the abstract syntax rules

$$C, D \rightarrow A \mid \forall R{:}\, C \mid \exists R{:}\, C \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \text{atleast } N\, R \mid \text{atmost } N\, R.$$

An **interpretation** $I = (\mathcal{D}^I, I[\bullet])$ consists of a set $\mathcal{D}^I$ (the **domain** of I) and a function $I[\bullet]$ (the **interpretation function** of I) that maps every concept description to a subset of $\mathcal{D}^I$, and every role symbol to a subset of $\mathcal{D}^I \times \mathcal{D}^I$, satisfying the following equations:

- $I[\top] = \mathcal{D}^I$
- $I[\bot] = \emptyset$
- $I[\forall R{:}\, C] = \{ a \in \mathcal{D}^I \mid \forall\, (a,b) \in I[R] : b \in I[C] \}$
- $I[\exists R{:}\, C] = \{ a \in \mathcal{D}^I \mid \exists\, (a,b) \in I[R] : b \in I[C] \}$

- $I[C \sqcap D] = I[C] \cap I[D]$
- $I[C \sqcup D] = I[C] \cup I[D]$
- $I[\neg C] = \mathcal{D}^I - I[C]$
- $I[\text{atleast N R}] = \{ a \in \mathcal{D}^I \mid \| \{ b \in \mathcal{D}^I \mid (a,b) \in I[R] \} \| \geq N \}$
- $I[\text{atmost N R}] = \{ a \in \mathcal{D}^I \mid \| \{ b \in \mathcal{D}^I \mid (a,b) \in I[R] \} \| \leq N \}$,

where $\|\bullet\|$ denotes the cardinality of sets.

An interpretation I is a **model** for a concept description C if $I[C]$ is nonempty. A concept description is **consistent** if it has a model. If C and D are concept descriptions, then C is **subsumed** by D if $I[C] \subseteq I[D]$ for every interpretation I, and C is **equivalent** to D if $I[C] = I[D]$ for every interpretation I.

Proposition 2.1. *Let C and D be concept descriptions. Then:*

a) C is subsumed by D if and only if $C \sqcap \neg D$ is inconsistent.

b) C is equivalent to D if and only if $(C \sqcap \neg D) \sqcup (D \sqcap \neg C)$ is inconsistent.

Proof. Follows immediately from the definitions. □

Since our language admits unions, intersections and complements an algorithm for checking consistency can be used for deciding subsumption and equivalence. In the following chapters we will work out an algorithm for checking consistency of $\mathcal{ALCN}$-concept descriptions and determine its complexity.

The syntax of $\mathcal{ALCN}$ is redundant. For instance, $\exists R: C$ is equivalent to $\neg\forall R: \neg C$, $\bot$ is equivalent to $A \sqcap \neg A$ for every concept symbol A, and $C \sqcup D$ is equivalent to $\neg(\neg C \sqcap \neg D)$.

The redundant syntax allows for the simplification of complex complements to

simple complements of the form $\neg A$, where A is a concept symbol different from $\top$ and $\bot$. This can be done by the following **simplification rules** reducing concept descriptions to equivalent concept descriptions:

- $\neg\top \rightarrow \bot$
- $\neg\bot \rightarrow \top$
- $\neg(\forall R: C) \rightarrow \exists R: \neg C$
- $\neg(\exists R: C) \rightarrow \forall R: \neg C$
- $\neg(C \sqcap D) \rightarrow \neg C \sqcup \neg D$
- $\neg(C \sqcup D) \rightarrow \neg C \sqcap \neg D$
- $\neg\neg C \rightarrow C$
- $\neg(\text{atleast N R}) \rightarrow \begin{cases} \text{atmost N-1 R} & \text{if } N > 0 \\ \bot & \text{if } N = 0 \end{cases}$
- $\neg(\text{atmost N R}) \rightarrow \text{atleast N+1 R}$.

A concept description is called **simple** if it contains only simple complements.

Proposition 2.2. *Let the concept description C´ be obtained from the concept description C by application of a simplification rule. Then :*

a) C is equivalent to C´,

b) C is consistent if and only if C´ is consistent.

Proof. It is easy to see that the simplification rules preserve consistency and inconsistency. □

Proposition 2.3. *For every concept description one can compute in linear time an equivalent simple concept description.*

Proof. A simple concept description can be obtained from a concept description in linear time by rewriting with the simplification rules in top-down order. □

2.2. Related ACD-languages

Schmidt-Schauß and Smolka [88] investigate different sublanguages of $\mathcal{ALCN}$. They define the following sublanguages:

language:	abstract syntax rule:
$\mathcal{AL}$	$C, D \rightarrow A \mid \forall R : C \mid \exists R: \top \mid C \sqcap D \mid \neg A$
$\mathcal{ALE}$	$C, D \rightarrow A \mid \forall R : C \mid \exists R: C \mid C \sqcap D \mid \neg A$
$\mathcal{ALU}$	$C, D \rightarrow A \mid \forall R : C \mid \exists R: \top \mid C \sqcap D \mid C \sqcup D \mid \neg A$
$\mathcal{ALC}$	$C, D \rightarrow A \mid \forall R : C \mid \exists R: C \mid C \sqcap D \mid C \sqcup D \mid \neg C$

The names of the languages are put together as follows: $\mathcal{ALE}$ is obtained from $\mathcal{AL}$ by adding general existential role quantifications, $\mathcal{ALU}$ is obtained from $\mathcal{AL}$ by adding unions, $\mathcal{ALC}$ is obtained from $\mathcal{AL}$ by adding general complements, and $\mathcal{ALCN}$ is obtained from $\mathcal{ALC}$ by adding the so-called *number restriction* operators (atleast N R) and (atmost N R).

In [Schmidt-Schauß/Smolka 88] we can find the following results:

- consistency of $\mathcal{AL}$ - concept descriptions can be checked in linear time
- inconsistency of $\mathcal{ALE}$ - concept descriptions can be decided in nondeterministic linear time
- checking consistency of $\mathcal{ALU}$ - concept descriptions is NP-complete
- checking consistency and subsumption of $\mathcal{ALC}$ - concept descriptions are PSPACE-complete problems that can be decided with linear space.

Checking consistency of $\mathcal{ALE}$ - concept descriptions is quite an interesting problem. Up to this day we could find neither a polynomial algorithm nor a proof for the NP-hardness for this problem in the literature.

Schmidt-Schauß [88] shows that an ACD-language with role value maps has an undecidable subsumption problem if roles are not restricted to partial functions, even if union and complement are not available.

3 Constraint systems

The applicative structure of concept descriptions is not very suitable for devising consistency checking algorithms. Therefore every concept description will be translated into a constraint system such that the concept description is consistent if and only if the constraint system is consistent. For constraint systems we will give transformation rules such that we obtain constraint systems, that can be checked in polynomial time for consistency. This fundamental technique has also been used successfully for Feature Logic [Smolka 88] and the ACD-languages $\mathcal{AL}$, $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{ALC}$ [Schmidt-Schauß/Smolka 88].

We assume the existence of two further disjoint alphabets of symbols, called **individual variables** V^I and **concept variables** V^C, respectively. The letters x, y, z will always range over individual variables and the letters X, Y, Z will always range over concept variables.

Let I be an interpretation. An **I-assignment** is a function α that maps every individual variable to an element of $\mathcal{D}^I$ and every concept variable to a subset of $\mathcal{D}^I$. ASS^I is the set of all I-assignments.

A **constraint** has one of the following forms:

$X \sqsubseteq C$, $X(\forall R)Y$, $X(\exists R)Y$, $X \sqsubseteq Y \sqcup Z$, $x:X$, $x:A$, xRy, $X(\text{atleast N R})$, $X(\text{atmost N R})$,

where the C in $X \sqsubseteq C$ is a simple concept description and the A in x:A is a concept symbol. Let I be an interpretation. The interpretation function $I[\bullet]$ of I will be extended to constraints by interpreting them as sets of I-assignments:

- $\mathcal{I}[X \sqsubseteq C] = \{ \alpha \in ASS^{\mathcal{I}} \mid \alpha(X) \subseteq \mathcal{I}[C] \}$
- $\mathcal{I}[X(\forall R)Y] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X)\ \forall (a,b) \in \mathcal{I}[R] : b \in \alpha(Y) \}$
- $\mathcal{I}[\exists(\forall R)Y] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X)\ \exists (a,b) \in \mathcal{I}[R] : b \in \alpha(Y) \}$
- $\mathcal{I}[X \sqsubseteq Y \sqcup Z] = \{ \alpha \in ASS^{\mathcal{I}} \mid \alpha(X) \subseteq \alpha(Y) \cup \alpha(Z) \}$
- $\mathcal{I}[x:X] = \{ \alpha \in ASS^{\mathcal{I}} \mid \alpha(x) \in \alpha(X) \}$
- $\mathcal{I}[x:A] = \{ \alpha \in ASS^{\mathcal{I}} \mid \alpha(x) \in \mathcal{I}[A] \}$
- $\mathcal{I}[xRy] = \{ \alpha \in ASS^{\mathcal{I}} \mid (\alpha(x),\alpha(y)) \in \mathcal{I}[R] \}$
- $\mathcal{I}[X(\text{atleast N R})] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X) : \| \{ b \in \mathcal{D}^{\mathcal{I}} \mid (a,b) \in \mathcal{I}[R] \} \| \geq N \}$
- $\mathcal{I}[X(\text{atmost N R})] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X) : \| \{ b \in \mathcal{D}^{\mathcal{I}} \mid (a,b) \in \mathcal{I}[R] \} \| \leq N \}$.

A **constraint system** S is a finite, nonempty set of constraints. The interpretation of a constraint system S by an interpretation $\mathcal{I}$ is defined as $\mathcal{I}[S] := \bigcap_{c \in S} \mathcal{I}[c]$. An interpretation $\mathcal{I}$ is a **model** for S if $\mathcal{I}[S]$ is nonempty. A constraint system is **consistent** if it has a model.

We obtain the **standard interpretation** $\mathcal{I}_s$ of a constraint system S by taking for $\mathcal{D}^{\mathcal{I}_s}$ all individual variables occurring in S, for $\mathcal{I}_s[A]$ all x such that x:A is in S, and by taking for $\mathcal{I}_s[R]$ all pairs (x,y) such that xRy is in S. The **standard $\mathcal{I}$-assignment** α_s is defined by mapping individual variables to themselves and by taking for $\alpha(X)$ all x such that x:X is in S.

The next proposition shows the relationship between simple $\mathcal{ALCN}$-concept descriptions and constraint systems.

Proposition 3.1. *Let x be an individual variable and X be a concept variable. A simple concept description C is consistent if and only if the constraint system {x:X, X ⊑ C} is consistent.*

Proof. "⇒": Suppose the simple concept description C is consistent. Then there exists an interpretation $I = (\mathcal{D}^I, I[\bullet])$ with $I[C] \neq \emptyset$. Let x be an individual variable, X be a concept variable, and S be the constraint system $\{x{:}X, X \sqsubseteq C\}$. We will define an I-assignment α such that $\alpha \in I[S]$.

Since $I[C] \neq \emptyset$, there exists a $d \in \mathcal{D}^I$ such that $d \in I[C]$. Define $\alpha(x) := d$ and $\alpha(X) := I[C]$, $\alpha(X) := \{d\}$. Then $\alpha \in I[S]$, and S is consistent.

"⇐": Suppose the constraint system $\{x{:}X, X \sqsubseteq C\}$ is consistent. Then there exists an I-assignment α with $\alpha(x) \in \alpha(X)$ and $\alpha(X) \subseteq I[C]$. Hence $\alpha(x) \in I[C]$ and C is consistent. □

A constraint system S is **simple** if for every constraint $X \sqsubseteq C$ in S the concept description C is either a concept symbol different from $\top$ and $\bot$, or a complemented concept symbol.

The following **unfolding rules** can be used to simplify general constraint systems to simple constraint systems:

- $X \sqsubseteq \forall R{:}\, C \rightarrow X(\forall R)Y,\ Y \sqsubseteq C$, where Y is a new concept variable
- $X \sqsubseteq \exists R{:}\, C \rightarrow X(\exists R)Y,\ Y \sqsubseteq C$, where Y is a new concept variable
- $X \sqsubseteq C \sqcap D \rightarrow X \sqsubseteq C, X \sqsubseteq D$
- $X \sqsubseteq C \sqcup D \rightarrow X \sqsubseteq Y \sqcup Z,\ Y \sqsubseteq C, Z \sqsubseteq D$, where Y, Z are new concept variables
- $X \sqsubseteq \top \rightarrow$ nothing
- $X \sqsubseteq \bot \rightarrow X \sqsubseteq A, X \sqsubseteq \neg A$, where A is a new concept symbol
- $X \sqsubseteq \text{atleast N R} \rightarrow \begin{cases} \text{nothing} & \text{if N} = 0 \\ X(\text{atleast N R}) & \text{if N} > 0 \end{cases}$
- $X \sqsubseteq \text{atmost N R} \rightarrow X(\text{atmost N R})$.

Proposition 3.2. *Let the constraint system S′ be obtained from the constraint system S by the application of an unfolding rule. Then S is consistent if and only if S′ is consistent.*

Proof. Suppose the constraint system S′ has been obtained from S by the second rule. Then $S = \{X \sqsubseteq \exists R: C\} \cup S_{rest}$ and $S' = \{X(\exists R)Y, Y \sqsubseteq C\} \cup S_{rest}$, where Y is a new concept variable not occurring in S.

If S is consistent, then there exists an $\mathcal{I}$-assignment α with $\alpha(X) \subseteq \mathcal{I}[\exists R: C]$. Put $\alpha'(x) = \alpha(x)$ for every $x \in V^I$, $\alpha'(X) = \alpha(X)$ for every $X \in V^C$, $X \neq Y$ and $\alpha'(Y) = \mathcal{I}[C]$. Then for every $a \in \alpha'(X)$ exists $(a,b) \in \mathcal{I}[R]$ with $b \in \alpha'(Y)$ and $\alpha'(Y) \subseteq \mathcal{I}[C]$. Thus $\alpha' \in \mathcal{I}[X(\exists R)Y, Y \sqsubseteq C]$ and $S' = \{X(\exists R)Y, Y \sqsubseteq C\} \cup S_{rest}$ is consistent.

If S′ is consistent, then there exists an $\mathcal{I}$-assignment α such that 1.) for every $d \in \alpha(X)$ there exists an $e \in \alpha(Y)$ such that $(d,e) \in \mathcal{I}[R]$ and 2.) $\alpha(Y) \subseteq \mathcal{I}[C]$. This implies that for every $d \in \alpha(X)$ there exists an $e \in \mathcal{I}[C]$ such that $(d,e) \in \mathcal{I}[R]$. Hence $\alpha \in \mathcal{I}[S]$ and S is consistent.

The proofs for the other rules are similar. □

Proposition 3.3. *For every constraint system S one can compute in linear time a simple constraint system S′ such that S is consistent if and only if S′ is consistent.*

Proof. A simple constraint system S′ can be obtained from a constraint system S in linear time by rewriting with the unfolding rules in top-down order. □

Let S be a simple constraint system obtained by unfolding the constraint $X \sqsubseteq C$ where C is a simple $\mathcal{ALCN}$- concept description. Then the constraint system $S' = S \cup \{x:X\}$ is called **fresh** and the individual variable x is called the **individual root** of S′. Note that the constraint x:X in S′ is the only one that contains an individual variable.

Theorem 3.4. *For every concept description C one can compute in linear time a fresh constraint system S such that C is consistent if and only if S is consistent.*

Proof. The concept description C is transformed into a simple concept description C′ using the simplification rules. Now the constraint system {x:X, X ⊑ C′} is created, which then is simplificated to a fresh constraint system using the unfolding rules. All three steps require at most linear time and preserve consistency and inconsistency. □

A simple constraint system defines a directed graph, called its **skeleton**, as follows: every concept variable occurring in the constraint system is taken as a node, the constraints X(∀R)Y and X(∃R)Y define universal and existential edges from X to Y, and a constraint X ⊑ Y ⊔ Z defines an or-connected pair of edges from X to Y and Z. Furthermore, the constraints X ⊑ A and X ⊑ ¬A define A and ¬A as labels of node X. The constraints X(atleast N R) and X(atmost N R) define (atleast N R) and (atmost N R) as labels of node X. Thus every node has a finite, possibly empty set of labels. The individual constraints x: X and xRy don't contribute to the skeleton. A **constraint tree** is a simple constraint system whose skeleton is a tree. Note that the skeleton of a fresh constraint system is a tree.

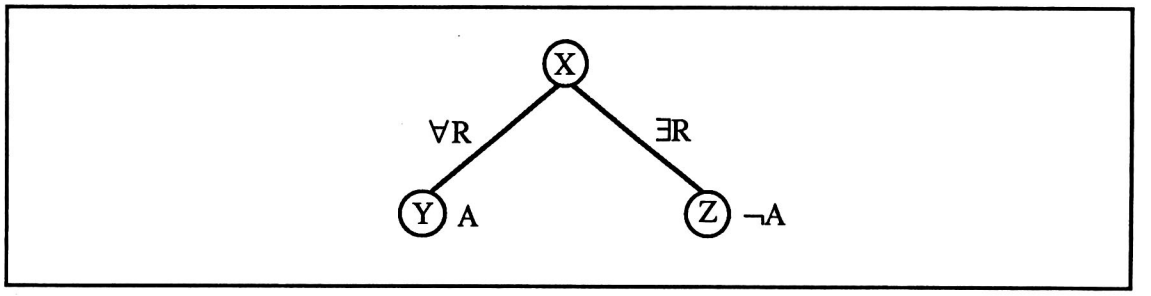

Figure 3.1. A constraint tree representing the constraint system
S = {x:X, X(∀R)Y, Y ⊑ A, X(∃R)Z, Z ⊑ ¬A}.

Example 3.1.

Let C = (∀R: A) ⊓ ¬(∀R: A) be a concept description. The simple concept description C′ = (∀R: A) ⊓ (∃R: ¬A) is obtained from C using the simplification rules. The constraint system {x:X, X ⊑ C′} is transformed into the fresh constraint system

$S = \{x{:}X, X(\forall R)Y, Y \sqsubseteq A, X(\exists R)Z, Z \sqsubseteq \neg A\}$ by application of the unfolding rules. A constraint tree for S is shown in figure 3.1.

4 Propagation

We now define so-called complete constraint systems whose consistency can be checked in polynomial time. Every fresh constraint system can be completed (by application of completion rules) to a complete constraint system preserving consistency and inconsistency by adding individual constraints of the form xRy, x:X and x:A, where A is either a concept symbol or a complemented concept symbol.

4.1. Completion rules

Let S be a constraint system. For an individual variable x we count the number of individual variables y with xRy for some role symbol R. We therefore define

$$n_{R,S}(x) := \|\{\, y \in V^I \mid xRy \in S \,\}\|.$$

If it is obvious which constraint system is considered, we often write $n_R(x)$ instead of $n_{R,S}(x)$. With

$$\{y \leftarrow z\}S$$

we define the constraint system that is obtained from S by replacing each occurrence of y by z.

Proposition 4.1.1. *Let S be a constraint system. Then:*

1. *if x:X, xRy and X(∀R)Y are in S, then S is consistent if and only if S ∪ {y:Y} is consistent*
2. *if x:X and X(∃R)Y are in S and y is an individual variable not occurring in S, then S is consistent if and only if S ∪ {xRy, y:Y} is consistent*
3. *if x:X and X ⊑ Y ⊔ Z are in S, then S is consistent if and only if S ∪ {x:Y} or S ∪ {x:Z} is consistent*

4. *if x:X and X ⊑ A are in S, then S is consistent if and only if S ∪ {x:A} is consistent*
5. *if x:X and X(atleast N R) are in S and N > $n_R(x)$ then S is consistent if and only if S ∪ {xRy} is consistent, where y is an individual variable not occurring in S*
6. *if x:X and X(atmost N R) are in S, N > 0 and $n_R(x)$ > N, then there is a choice of y, z ∈ V^I with xRy and xRz in S, such that {y ← z}S is consistent if and only if S is consistent.*
7. *if x:X, X(atleast N R), x:Y, Y(atmost M R) are in S and N > M, then S is not consistent*
8. *if x:X, xRy and X(atmost 0 R) are in S, then S is not consistent.*

Proof. We show the first and seventh part of the proposition. The proofs of the other parts are similar.

1. Let S = {x:X, xRy, X(∀R)Y} ∪ S_{rest}.

Suppose S is consistent. Then there exists an $\mathcal{I}$-assignment α with α(x) ∈ α(X), (α(x),α(y)) ∈ $\mathcal{I}$[R], and for every a ∈ α(X) such that (a,b) ∈ $\mathcal{I}$[R] we have b ∈ α(Y). It follows that α(y) ∈ α(Y), and therefore S′ = S ∪ {y:Y} is consistent.

Now suppose S is not consistent. Then $\mathcal{I}$[S] = Ø and $\mathcal{I}$[S′] = $\mathcal{I}$[S] ∩ $\mathcal{I}$[y:Y] = Ø. Thus S′ is not consistent.

7. Let {x:X, X(atleast N R), x:Y, Y(atmost M R)} ⊆ S and N > M. Suppose there exists an $\mathcal{I}$-assignment α with α ∈ $\mathcal{I}$[x:X, X(atleast N R), x:Y, Y(atmost M R)]. Then we have ||{b ∈ $\mathcal{D}^I$ | (α(x),b) ∈ $\mathcal{I}$[R]}|| ≥ N and ||{b ∈ $\mathcal{D}^I$ | (α(x),b) ∈ $\mathcal{I}$[R]}|| ≤ M. Since N > M this is a contradiction. Thus $\mathcal{I}$[x:X, X(atleast N R), x:Y, Y(atmost M R)] = Ø and S is not consistent. □

Every constraint system can be extended using the following **R_1-completion rules**:

1. S $\rightarrow_\forall$ {y:Y} ∪ S

   if x:X, xRy and X(∀R)Y are in S and y:Y is not in S

2. $S \rightarrow_{\exists} \{y:Y, xRy\} \cup S$

   if $x:X$ and $X(\exists R)Y$ are in S and there exists no individual variable z such that xRz and $z:Y$ are in S, and y is an individual variable not occurring in S

3. $S \rightarrow_{\sqcup} \{x:Z\} \cup S$

   if $x:X$ and $X \sqsubseteq Y_1 \sqcup Y_2$ are in S, neither $x:Y_1$ nor $x:Y_2$ is in S, and Z is either Y_1 or Y_2

4. $S \rightarrow_{\sqsubseteq} \{x:A\} \cup S$

   if $x:X$ and $X \sqsubseteq A$ are in S and $x:A$ is not in S

5. $S \rightarrow_{atleast} \{xRy\} \cup S$

   if $x:X$, $X(\text{atleast N R})$ are in S and $n_R(x) < N$, and y is a new individual variable not occurring in S

6. $S \rightarrow_{atmost} \{y \leftarrow z\}S$

   if $x:X$, $X(\text{atmost N R})$, xRy and xRz are in S and $n_R(x) > N$

7. $S \rightarrow_{error} \{x:A, x:\neg A\}$

   if $x:X$, $X(\text{atleast N R})$, $x:Y$, $Y(\text{atmost M R})$ are in S and $N > M$ or

   if $x:X$, $X(\text{atmost 0 R})$, xRy are in S.

A constraint system is **R_1-complete** if no R_1-completion rule applies to it.

Proposition 4.1.2. *Let S and S′ be constraint systems. Then:*

a) If S′ is obtained from S by application of the (deterministic) $\rightarrow_{\forall}$ - rule, $\rightarrow_{\exists}$ - rule, $\rightarrow_{\sqsubseteq}$ - rule, $\rightarrow_{atleast}$ - rule or $\rightarrow_{error}$ - rule, then S is consistent if and only if S′ is consistent.

b) If S′ is obtained from S by application of the (nondeterministic) $\rightarrow_{\sqcup}$ - rule or $\rightarrow_{atmost}$ - rule, then S is consistent if S′ is consistent. Furthermore there is a choice for S′ such that S′ is consistent if and only if S is consistent.

Proof. Follows from proposition 4.1.1. □

4.2. Consistency checking as completion

The next proposition justifies our interest in R_1-complete constraint systems.

Proposition 4.2.1. *An R_1-complete constraint system is inconsistent if and only if it contains, for some $x \in V^I$ and some concept symbol A, the constraints x:A and x:¬A.*

Proof. "⇒": Let S be an R_1-complete constraint system not containing the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A. We show that the standard interpretation I_S of S is a model for S. In particular we show that for the standard I-assignment α_S we have $\alpha_S \in I_S[c]$ for every $c \in S$ and hence $\alpha_S \in I_S[S]$. We distinguish three kinds of constraints: Constraints of the form x:X, x:A, $X \sqsubseteq A$, xRy, constraints of the $X(\forall R)Y$, $X(\exists R)Y$, $X \sqsubseteq Y \sqcup Z$, X(atleast N R), X(atmost N R), and constraints of the form x:¬A.

1. We show that $\alpha_S \in I_S[x:X]$: Because of the definition of I_S and α_S we have $\alpha_S(x) = x$ and $\alpha_S(X) = \{x \mid x:X \in S\}$. Therefore $\alpha_S(x) \in \alpha_S(X)$ and $\alpha_S \in I_S[x:X]$. Similarly, it can be shown that $\alpha_S \in I_S[x:A]$, $\alpha_S \in I_S[X \sqsubseteq A]$ and $\alpha_S \in I_S[xRy]$.

2. Next we show that $\alpha_S \in I_S[X(\exists R)Y]$. If there is no constraint x:X in S, then it is easy to see that $\alpha_S \in I_S[X(\exists R)Y]$. Now assume that $\{x:X, X(\exists R)Y\} \subseteq S$. Since S is R_1-complete, there are constraints xRy and y:Y in S, since the $\rightarrow_\exists$ - rule doesn't apply. Hence $\alpha_S \in I_S[\{ x:X, X(\exists R)Y \}]$. Similarly, it can be shown that $\alpha_S \in I_S[X(\forall R)Y]$, $\alpha_S \in I_S[X \sqsubseteq Y \sqcup Z]$, $\alpha_S \in I_S[X(\text{atleast N R})]$ and $\alpha_S \in I_S[X(\text{atmost N R})]$.

3. We prove that $\alpha_S \in I_S[x:\neg A]$ if the constraint x:A is not in S. Suppose $x:A \notin S$. Then $\alpha_S(x) \notin I_S[A]$ and $\alpha_S(x) \in \mathcal{D}^{I_S} - I_S[A]$. Hence $\alpha_S \in I_S[x:\neg A]$.

"⇐": Suppose the R_1-complete constraint system S contains the constraints x:A and x:¬A for some $x \in V^I$ and some concept symbol A. Since the sets $I[A]$ and $I[\neg A]$ are disjoint for every interpretation I, there doesn't exist an I-assignment α that maps x simultaneously to an element of $I[A]$ and $I[\neg A]$. Thus $I[x:A] \cap I[x:\neg A] = \emptyset$. Since

$\{x{:}A, x{:}\neg A\} \subseteq S$ we conclude that $\mathcal{I}[S]$ is empty and S is inconsistent. □

Example 4.2.1.

Let S = { x:X, X($\forall$R)Y, Y $\sqsubseteq$ A, X($\exists$R)Z, Z $\sqsubseteq$ $\neg$A } be a fresh constraint system (see example 3.1.). The following constraint systems are created using the R_1-completion rules:

$$S \rightarrow_{\exists} S \cup \{xRy, y{:}Z\}$$
$$\rightarrow_{\sqsubseteq} S \cup \{xRy, y{:}Z, y{:}\neg A\}$$
$$\rightarrow_{\forall} S \cup \{xRy, y{:}Z, y{:}\neg A, y{:}Y\}$$
$$\rightarrow_{\sqsubseteq} S \cup \{xRy, y{:}Z, y{:}\neg A, y{:}Y, y{:}A\} = S'$$

Observe that only deterministic completion rules apply to S. The R_1-complete constraint system S´ is not consistent since S´ contains the constraints y:A and y:$\neg$A.

4.3. A completion procedure

Let S be a fresh constraint system. In the following we will show, that in finitely many propagation steps one can nondeterministically compute an R_1-complete constraint system S´, such that S is consistent if and only if S´ is consistent. To do this, we have to impose some control on the application of the rules in order to avoid infinite chains of completion steps. The following example illustrates that such a control is indeed necessary.

Example 4.3.1.

Let S = {x:X, X(atleast 2 R), X(atmost 1 R)} be a fresh constraint system. Then:

$S \rightarrow_{atleast} S \cup \{xRy\} \rightarrow_{atleast} S \cup \{xRy, xRz\} \rightarrow_{atmost} \{z \leftarrow y\}S = S \cup \{xRy\} \rightarrow_{atleast}$ $S \cup \{xRy, xRz\} \rightarrow_{atmost}$ ... and we have an infinite chain.

On the other hand if the control strategy applies the $\rightarrow_{error}$ - rule before applying the $\rightarrow_{most}$ - rule, we get $S \rightarrow_{atleast} S \cup \{xRy\} \rightarrow_{atleast} S \cup \{xRy, xRz\} \rightarrow_{error} \{x{:}A, x{:}\neg A\}$,

and we have an R_1-complete constraint system.

Let S be a constraint system. We write $S \Rightarrow_* S'$, $* \in \{\forall, \exists, \sqcup, \sqsubseteq$, atleast, atmost, error$\}$, if $S = S'$ or the constraint system S' is obtained from S by applications of the $\rightarrow_*$ - rule, and the $\rightarrow_*$ - rule doesn't apply to S'.

Let S be a constraint system whose skeleton is a constraint tree. We assign **levels** to the concept variables occurring in S as follows:

1. the concept variable that is the root of the constraint tree has the level 0,
2. if the constraint tree contains a constraint $X(\forall R)Y$ or $X(\exists R)Y$ and X has the level n, then Y has the level n+1,
3. if the constraint tree contains a constraint $X \sqsubseteq Y \sqcup Z$, then X, Y and Z all have the same level.

This defines a unique level assignment for constraint systems that are obtained from fresh constraint systems by application of completion rules. Remember that the skeleton of a fresh constraint system is a tree.

We extend the level assignment to constraints: A **constraint** is **at level** n, if the first concept variable occuring in the constraint has level n. Constraints of the form xRy will not be assigned a level. A constraint system is **complete at level** n, if no completion rule applies to constraints at level n.

Next we define the recursive function *Completion*. The call *Completion(S,0)* computes an R_1-complete constraint system for the constraint system S given as argument. The algorithm is obtained from the R_1-completion rules by adding some control. The control guarantees that a "breadth-first completion" is performed: As long as possible apply the R_1-completion rules to constraints at level i, and then apply the rules to constraints at level i+1. The number given as second argument indicates the level. The

function terminates, since there exists a number n bounding the levels of constraints in S. Suppose *Completion(S,i)* is called. Let S_i be the set of all constraints at level i. The sequence of applications of the R_1-completion rules is important: Since the $\rightarrow_{\sqcup}$ - rule is the only one that introduces constraints at level i, this rule is applied first. The $\rightarrow_{\exists}$ - rule and the $\rightarrow_{atleast}$ - rule add constraints of the form xRy, therefore these rules are applied before the $\rightarrow_{\forall}$ - rule. Furthermore, the $\rightarrow_{error}$ - rule must apply before the $\rightarrow_{atmost}$ - rule to avoid infinite chains of completion steps (see example 4.3.1.).

Function **Completion** (S, i)

let S_i be the set of all constraints at level i in S

if $S_i = \emptyset$

then return S

else let $S_{\sqcup}$, $S_{\exists}$, $S_{atleast}$, $S_{\forall}$, $S_{\sqsubseteq}$, S_{error}, $S_{i\text{-complete}}$ such that

$S_i \Rightarrow_{\sqcup} S_{\sqcup} \Rightarrow_{\exists} S_{\exists} \Rightarrow_{atleast} S_{atleast} \Rightarrow_{\forall} S_{\forall} \Rightarrow_{\sqsubseteq} S_{\sqsubseteq} \Rightarrow_{error}$

$S_{error} \Rightarrow_{atmost} S_{i\text{-complete}}$;

Completion ($S \cup S_{i\text{-complete}}$, i+1)

end Completion.

Proposition 4.3.1. *Let S be a constraint system obtained from an $\mathcal{ALCN}$ - concept description. Consider the call Completion(S,0). Let*

$S^j = S \cup S_{0\text{-complete}} \cup S_{1\text{-complete}} \cup \ldots \cup S_{(j-1)\text{-complete}}$,

that is, S^j is the argument of the j-th recursive call. Then:

a) The constraint system $S_{i\text{-complete}}$ is complete at level i for all i.

b) The constraint system S^j is complete at level i for all $i < j$.

c) The call Completion(S,0) terminates.

Proof: a) Let S_i be the set of all constraints at level i. Observe that the constraint systems $S_{\sqcup}$, $S_{\exists}$, $S_{atleast}$, $S_{\forall}$, $S_{\sqsubseteq}$, S_{error} and $S_{i\text{-complete}}$ have the following properties:

$S_{\sqcup}$ is obtained from S_i by adding the constraints x:Y or x:Z for every $\{x:X, X \sqsubseteq Y \sqcup Z\} \subseteq S_i$. The constraints x:Y and x:Z are at level i.

$S_{\exists}$ is obtained from $S_{\sqcup}$ by adding the constraints xRy and y:Y for every $\{x:X, X(\exists R)Y\} \subseteq S_{\sqcup}$. The constraint y:Y is at level i+1.

$S_{atleast}$ is obtained from $S_{\exists}$ by adding constraints of the form xRy for every $\{x:X, X(\text{atleast } N\ R)\} \subseteq S_{\exists}$, such that $N = \|\{ y \in V^I \mid xRy \in S_{atleast}\}\|$.

$S_{\forall}$ is obtained from $S_{atleast}$ by adding the constraints y:Y for every $\{x:X, X(\forall R)Y, xRy\} \subseteq S_{atleast}$. The constraint y:Y is at level i+1.

$S_{\sqsubseteq}$ is obtained from $S_{\forall}$ by adding the constraints x:A for every $\{x:X, X \sqsubseteq A\} \subseteq S_{\forall}$. If the $\rightarrow_{error}$ - rule applies to $S_{\sqsubseteq}$, then $S_{error} = \{x:A, x:\neg A\}$, otherwise $S_{error} = S_{\sqsubseteq}$. Finally the constraint system $S_{i\text{-complete}}$ is obtained from S_{error} by finitely many applications of the $\rightarrow_{atmost}$ - rule.

Now we will show that no R_1-completion rule applies to constraints at level i in $S_{i\text{-complete}}$.

Consider the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule:
Constraints of the form x:X, which have been introduced by the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\forall}$ - rule, are at level i+1. If the constraint x:X is at level i in $S_{i\text{-complete}}$, then x:X is already in $S_{\sqcup}$. Therefore the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule don't apply to $S_{i\text{-complete}}$.

Consider the $\rightarrow_{\forall}$ - rule:
Since the $\rightarrow_{\sqsubseteq}$ - rule, the $\rightarrow_{error}$ - rule and the $\rightarrow_{atmost}$ - rule, which are applied after the $\rightarrow_{\forall}$ - rule, don't introduce constraints of the form xRy and x:X, the $\rightarrow_{\forall}$ - rule doesn't apply to $S_{i\text{-complete}}$.

Consider the $\rightarrow_{atleast}$ - rule:
Assume that the $\rightarrow_{atleast}$ - rule applies to $S_{i\text{-complete}}$. Then
$\{x:X, X(\text{atleast } N\ R)\} \subseteq S_{i\text{-complete}}$ and $N > \|\{ y \mid xRy \in S_{i\text{-complete}}\}\|$. Since $N = \|\{ y \mid xRy \in S_{atleast}\}\|$ after the application of the $\rightarrow_{atleast}$ - rule, the constraints x:Y

and Y(atmost M R) must be in $S_{i\text{-complete}}$ and N > M. But then the $\rightarrow_{error}$ - rule would have been applied to $S_{\sqsubseteq}$ and we have a contradiction. Thus our assumption is false and the $\rightarrow_{atleast}$ - rule doesn't apply to $S_{i\text{-complete}}$.

It is obvious, that the $\rightarrow_{error}$ - rule and the $\rightarrow_{atmost}$ - rule don't apply to $S(i)_{complete}$. Now we have shown that no R_1-completion rule applies to constraints at level i in $S_{i\text{-complete}}$ and hence $S_{i\text{-complete}}$ is complete at level i.

b) We prove the claim by induction on the level i:

Base case i = 1: The constraint system $S^1 = S^0 \cup S_{0\text{-complete}}$ is complete at level 0.

Induction step: $S^{i+1} = S^i \cup S_{i\text{-complete}}$. The constraint system $S_{i\text{-complete}}$ is complete at level i and doesn't contain constraints at level l for l < i. By the induction hypothesis we know that S^i is complete at level l for l < i. Since $S_{i\text{-complete}}$ contains every constraint, which is in S^i and is at level i, we conclude that S^{i+1} is complete at level l for l < i+1.

c) Every constraint system obtained from a finite $\mathcal{ALCN}$ - concept description has a maximal level and hence the function terminates. □

The next theorem shows how to obtain an algorithm for checking consistency of $\mathcal{ALCN}$ - concept descriptions.

Theorem 4.3.2. *Let S be a fresh constraint system obtained from an $\mathcal{ALCN}$ concept description C.*

a) If C is consistent, then there exists a computation using Completion, such that the call Completion(S,0) returns an R_1-complete constraint system not containing x:A and x:¬A for any individual variable x and any concept symbol A.

b) If C is not consistent, then every call Completion(S,0) returns an R_1-complete constraint system containing x:A and x:¬A for some individual variable x and some concept symbol A.

Proof. Let C be an $\mathcal{ALCN}$- concept description. Then C is transformed into a fresh constraint system S using the simplification and unfolding rules. Note that C is consistent if and only if S is consistent. The call *Completion(S,0)* nondeterministically computes an R_1-complete constraint system S´ for S (proposition 4.3.1.). Furthermore there is a choice when applying the R_1-completion rules, such that S is consistent if and only if S´ is consistent (proposition 4.1.2). Hence, if every R_1-complete constraint system for S is inconsistent, then the concept description C is inconsistent, otherwise C is consistent. □

Now let us turn to the time and space complexity of this algorithm. Transforming a concept description C into a fresh constraint system can be done easily, that means in time O(n), where n is the length of C (see theorem 3.4.). Whether a constraint system contains the constraints x:A and x:¬A, can be checked in time $O(n^2)$, where n is the number of constraints.The following examples illustrate the complexity of the R_1-completion rules.

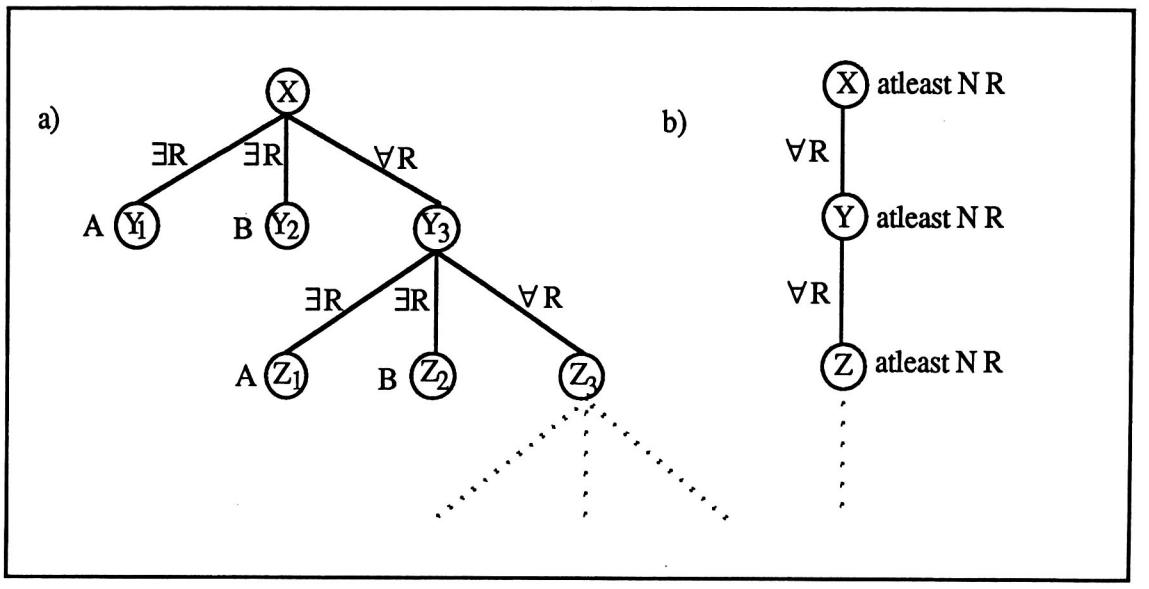

Figure 4.3.1. Constraint trees for constraint systems, where at least exponentially many propagation steps are necessary for a completion.

Example 4.3.2.

a) Let C = ∃R:A ⊓ ∃R:B ⊓ (∀R: (∃R:A ⊓ ∃R:B ⊓ ∀R: (...))) be a concept

description and let S be a fresh constraint system obtained from C (cf. [Schmidt-Schauß Smolka 88, page 17]). Figure 4.3.1.a shows a constraint tree for S. Every R_1-complete constraint system for S contains two constraints of the form y:Y at level 1, four constraints of the form y:Y at level 2, and in general 2^n constraints of the form y:Y at level n.

b) Figure 4.3.1.b shows a constraint tree for the fresh constraint system $S = \{x:X\} \cup S'$ where S' is obtained by unfolding the constraint $X \sqsubseteq C$ with $C = (\text{atleast } N\ R) \sqcap (\forall R: ((\text{atleast } N\ R) \sqcap \forall R: (\ldots)))$. Every R_1-complete constraint system of S contains N constraints of the form y:Y at level 1, N^2 constraints of the form y:Y at level 2, and in general N^n constraints of the form y:Y at level n.

In the following we investigate how many propagation steps are necessary to obtain an R_1-complete constraint system from a fresh constraint system. To ease our notation we make the following definition: With $\| c \in S \|$ we denote the number of all constraints of the form c, which are in the constraint system S.

Theorem 4.3.3. *If S is a fresh constraint system, then in at most exponentially many progagation steps one can nondeterministically compute an R_1-complete constraint system S', such that S is consistent if and only if S' is consistent.*

Proof: Let S be a fresh constraint system. The call *Completion(S,0)* computes nondeterministically an R_1-complete constraint system. There is a choice when applying the nondeterministic $\rightarrow_{\sqcup}$ - rule and $\rightarrow_{atmost}$ - rule, such that S is consistent if and only if the R_1-complete constraint system is consistent (theorem 4.3.2.).

Next we give an upper bound for the number of propagation steps needed to obtain an R_1-complete constraint system from a fresh constraint system:
Let S be a fresh constraint system with $\| S \| = m$, that is, m is the length of the string S. Consider the constraint system

$S^i := S \cup S_{0\text{-complete}} \cup S_{1\text{-complete}} \cup \ldots \cup S_{(i-1)\text{-complete}}$

that is computed by the call *Completion(S,0)*. Note that the constraint system S^i is complete at level j for $j < i$. Let S_i be the set of all constraints at level i in S^i. Put $s := \| S_i \|$. Remember that we obtain the constraint system $S_{i\text{-complete}}$ from S_i in the following manner:

$S_i \Rightarrow_{\sqcup} S_{\sqcup} \Rightarrow_{\exists} S_{\exists} \Rightarrow_{atleast} S_{atleast} \Rightarrow_{\forall} S_{\forall} \Rightarrow_{\sqsubseteq} S_{\sqsubseteq} \Rightarrow_{error} S_{error} \Rightarrow_{atmost} S_{i\text{-complete}}$.

The $\rightarrow_{\sqcup}$ - rule applies to every constraint of the form x:X at most $\| X \sqsubseteq Y \sqcup Z \in S_i \|$ times. Because $\| x:X \in S_i \| \leq s$ and $\| X \sqsubseteq Y \sqcup Z \in S_i \| \leq m$, the $\rightarrow_{\sqcup}$ - rule applies at most $(s * m)$ times. Each application of the $\rightarrow_{\sqcup}$ - rule adds one constraint of the form x:X, such that the constraint system $S_{\sqcup}$ contains at most $(s + s * m) \leq (2 * s * m)$ constraints of the form x:X.

The $\rightarrow_{\exists}$ - rule applies to every constraint of the form x:X at most $\| X(\exists R)Y \in S_{\sqcup} \|$ times. Since $\| x:X \in S_{\sqcup} \| \leq (2 * s * m)$ and $\| X(\exists R)Y \in S_{\sqcup} \| \leq m$, we know that the $\rightarrow_{\exists}$ - rule applies at most $(2 * s * m^2)$ times. Each application of the $\rightarrow_{\exists}$ - rule adds one constraint of the form x:X, such that the constraint system $S_{\exists}$ contains at most $(2 * s * m) + (2 * s * m^2) \leq (4 * s * m^2)$ constraints of the form x:X. Furthermore we note that for every individual variable x with $x:X \in S_{\exists}$ there are at most $\| X(\exists R)Y \in S_{\sqcup} \| \leq m$ constraints of the form xRy in the constraint system $S_{\exists}$ for some role symbol R and some individual variable y.

The $\rightarrow_{atleast}$ - rule applies to every constraint of the form x:X at most N times, where $N = \max \{ M \mid X(\text{atleast } M\ R) \in S_{\exists} \}$ for some role symbol R. Since the positive integer N is represented in binary, we have $\log_2(N) \leq m$ and hence N is not greater than 2^m. Thus the $\rightarrow_{atleast}$ - rule applies for x:X at most 2^m times. Because $\| x:X \in S_{\exists} \| \leq (4 * s * m^2)$, the $\rightarrow_{atleast}$ - rule applies at most $(4 * s * m^2 * 2^m)$ times. Note that the $\rightarrow_{atleast}$ - rule doesn't add constraints of the form x:X, hence $\| x:X \in S_{atleast} \| = \| x:X \in S_{\exists} \| \leq (4 * s * m^2)$. Furthermore the constraint system $S_{atleast}$ contains for every individual variable x with $x:X \in S_{atleast}$ at most 2^m constraints of the

form xRy for some role symbol R and some individual variable y.

The $\rightarrow_{\forall}$ - rule applies to the constraints x:X, xRy at most $\| X(\forall R)Y \in S_{atleast} \|$ times. Since we know that to every individual variable x with $x:X \in S_{atleast}$ there are at most 2^m constraints of the form xRy in the constraint system $S_{atleast}$ for some role symbol R and some individual variable y, and $\| X(\forall R)Y \in S_{atleast} \| \leq m$, the $\rightarrow_{\forall}$ - rule applies at most $\| x:X \in S_{atleast} \| * 2^m * \| X(\forall R)Y \in S_{atleast} \| \leq (4 * s * m^3 * 2^m)$ times. Each application of the $\rightarrow_{\forall}$ - rule adds one constraint of the form x:X, such that there are at most $(4 * s * m^2) + (4 * s * m^3 * 2^m) \leq (8 * s * m^3 * 2^m)$ constraints of the form x:X in the constraint system $S_{\forall}$.

The $\rightarrow_{\sqsubseteq}$ - rule applies to every constraint of the form x:X at most $\| X \sqsubseteq A \in S_{\forall} \|$ times. Because $\| x:X \in S_{\forall} \| \leq (8 * s * m^3 * 2^m)$ and $\| X \sqsubseteq A \in S_{\forall} \| \leq m$, the $\rightarrow_{\sqsubseteq}$ - rule applies at most $(8 * s * m^4 * 2^m)$ times.

The $\rightarrow_{error}$ - rule applies at most one time. If the $\rightarrow_{error}$ - rule doesn't apply, then $S_{error} = S_{\sqsubseteq}$.

Now let us consider the $\rightarrow_{atmost}$ - rule: We know that for every individual variable x with $x:X \in S_{\sqsubseteq}$ there are at most 2^m constraints of the form xRy in the constraint system $S_{\sqsubseteq}$ for some role symbol R and some individual variable y. Suppose (the worst case) X(atmost 1 R) $\in S_{\sqsubseteq}$. Then the $\rightarrow_{atmost}$ - rule applies to $x:X \in S_{\sqsubseteq}$ at most 2^m times. Since $\| x:X \in S_{\sqsubseteq} \| \leq (8 * s * m^3 * 2^m)$, the $\rightarrow_{atmost}$ - rule applies at most $(8 * s * m^3 * 2^m) * 2^m = (8 * s * m^3 * 2^{2m})$ times.

Altogether in at most

$$(s * m) + (2 * s * m^2) + (4 * s * m^2 * 2^m) + (4 * s * m^3 * 2^m) + (8 * s * m^4 * 2^m) + (8 * s * m^3 * 2^{2m})$$

$$= s * [m + (2 * m^2) + (4 * m^2 * 2^m) + (4 * m^3 * 2^m) + (8 * m^4 * 2^m) + (8 * m^3 * 2^{2m})]$$

$$\leq \| S_i \| * (27 * m^4 * 2^{2m})$$

propagation steps one can nondeterministically compute the constraint system

$S_{i\text{-complete}}$ from the constraint system S_i. Furthermore we know that

$$\| S_{i\text{-complete}} \| \leq \| S_i \| + 2 * \| S_i \| * (27 * m^4 * 2^{2m})$$
$$= \| S_i \| * (1 + 54 * m^4 * 2^{2m})$$

since an application of an R_1-completion rule adds at most two new constraints.

Now we compare the size of S^{i+1} to the size of S^i:

$$\begin{aligned}
\| S^{i+1} \| &= \| S^i \cup S_{i\text{-complete}} \| \\
&\leq \| S^i \| + \| S_{i\text{-complete}} \| \\
&\leq \| S^i \| + \| S_i \| * (1 + 54 * m^4 * 2^{2m}) \\
&\leq \| S^i \| + \| S^i \| * (1 + 54 * m^4 * 2^{2m}) \\
&\leq \| S^i \| * p(m) \qquad \text{where } p(m) \leq (55 * m^4 * 2^{2m}).
\end{aligned}$$

Note that a constraint system S with $\| S \| = m$ doesn't contain constraints at level M with $M \geq m$. Thus the constraint system S^m is R_1-complete and we have:

$$\begin{aligned}
\| S^m \| &\leq p(m) * \| S^{m-1} \| \\
&\leq p(m) * p(m) * \| S^{m-2} \| \leq .... \\
&\leq [p(m)]^m * \| S^0 \| \\
&= [p(m)]^m * m \\
&\leq [p(m)]^{m+1} \\
&= e^{(m+1) * \ln(p(m))} \\
&\leq e^{(m+1) * p'(m)} \quad \text{where } p'(m) = \ln(p(m)) \\
&\qquad \leq \ln(55) + 4 * \ln(m) + 2 * m * \ln(2).
\end{aligned}$$

If S is a fresh constraint system, then we obtain an R_1-complete constraint system from S by adding at most exponentially many constraints. Since every R_1-completion rule adds at most two constraints, we conclude that in at most exponentially many propagation steps one can nondeterministically compute an R_1-complete constraint system. □

4.4. Optimization

As seen in example 4.3.2. and theorem 4.3.3. in the worst case exponentially many propagation steps are necessary to obtain an R_1-complete constraint system for a fresh constraint system. One point to reduce this complexity is a modification of the $\rightarrow_{atleast}$ - rule. The idea behind this modification is as follows: Suppose a constraint system contains the constraints x:X and X(atleast N R). Then it is not really necessary to apply the $\rightarrow_{atleast}$ - rule N times adding the constraints $xRy_1, \ldots, xRy_N$. In the following we will show that it is sufficient to add only the single constraint xRy, where y represents the individual variables $y_1, \ldots, y_N$.

First we define the **R_2-completion rules**: We get the R_2-completion rules from the R_1-completion rules by modifying the $\rightarrow_{atleast}$ - rule in the following manner:
Let S be a constraint system. Then:

$S \rightarrow_{atleast} \{xRy\} \cup S$

if x:X, X(atleast N R) are in S, $N > 0$, $n_{R,S}(x) = 0$ and y is a new individual variable not occurring in S.

A constraint system is **R_2-complete** if no R_2-completion rule applies to it.

It is easy to see that the application of the modified $\rightarrow_{atleast}$ - rule to a constraint system preserves consistency and inconsistency. Furthermore we note that every R_1-complete constraint system is R_2-complete. An R_2-complete constraint system S is R_1-complete if it doesn't contain constraints of the form x:X and X(atleast N R) with $n_{R,S}(x) < N$.

In the following we consider constraint systems that are constraint trees. This assumption is justified because every constraint system obtained from a fresh constraint

system is a constraint tree. We will show that an R_2-complete constraint tree is consistent if and only if it doesn't contain the constraints x:A and x:$\neg$A for any x$\in V^I$ and any concept symbol A. To show this we pursue the following idea: Suppose the constraint tree S is R_2-complete and doesn't contain constraints of the form x:A and x:$\neg$A. Then we add constraints to S, such that we obtain an R_1-complete constraint tree S´ not containing constraints of the form x:A and x:$\neg$A. With proposition 4.2.1. – an R_1-complete constraint system is consistent if and only if it doesn't contain the constraints x:A and x:$\neg$A for any x$\in V^I$ and any concept symbol A – we know that S´ is consistent and we conclude that the R_2-complete constraint tree S is consistent. The following example illustrates how one can obtain an R_1-complete constraint tree from an R_2-complete constraint tree.

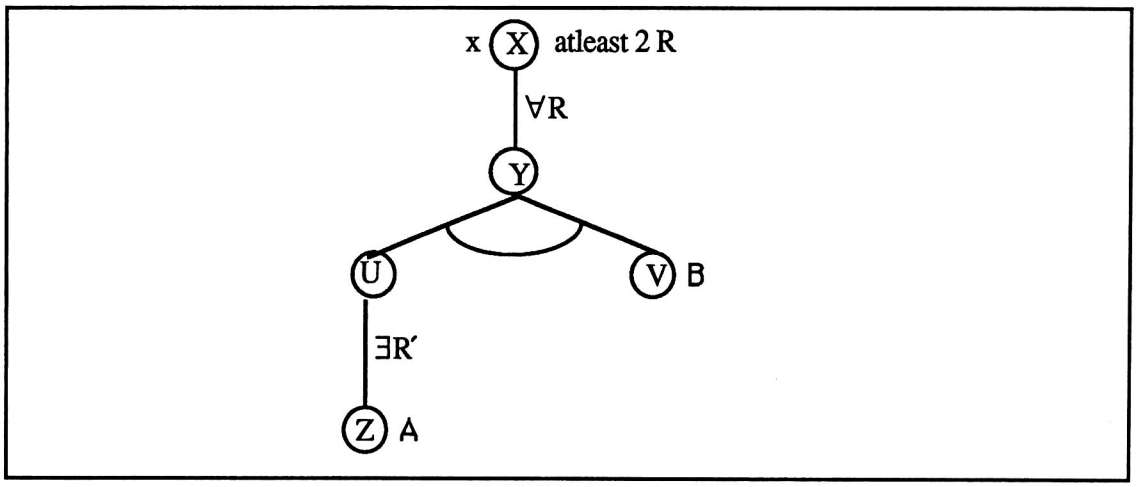

Figure 4.4.1. A constraint tree.

Example 4.4.1.

Let S = {x:X, X(atleast 2 R), X(∀R)Y, Y ⊑ U ⊔ V, V ⊑ B, U(∃R´)Z, Z ⊑ A} be a fresh constraint tree, which is shown in figure 4.4.1. We obtain the R_2-complete constraint tree S_2 from S in the following way:

$S \rightarrow_{atleast} S \cup \{xRy\}$

$\rightarrow_{\forall} \quad S \cup \{xRy, y:Y\}$

$\rightarrow_{\sqcup}$ $S \cup \{xRy, y:Y, y:U\}$

$\rightarrow_{\exists}$ $S \cup \{xRy, y:Y, y:U, yR'z, z:Z\}$

$\rightarrow_{\sqsubseteq}$ $S \cup \{xRy, y:Y, y:U, yR'z, z:Z, z:A\} = S_2$.

Note that the constraint tree S_2 doesn't contain constraints of the form x:A and x:$\neg$A. Now we construct the R_1-complete constraint tree S_1 which has the following properties:

- $S_2 \subseteq S_1$
- S_1 doesn't contain constraints of the form x:A and x:$\neg$A.

Since the constraint tree S_2 is not R_1-complete, we have to add constraints to S_2 to obtain the R_1-complete constraint tree S_1:

$S_1 = S_2 \cup \{xRy_2\}$ Now we have $\|\{ y \in V^I \mid xRy \in S_1 \}\| = 2$ and the $\rightarrow_{atleast}$ - rule of the R_1-completion rules doesn't apply to x:X, X(atleast 2 R).

$\cup \{y_2:Y\}$ Now the $\rightarrow_{\forall}$ - rule doesn't apply to x:X, xRy_2, X($\forall$R)Y.

$\cup \{y_2:U\}$ We have to add either the constraint y_2:U or y_2:V such that the $\rightarrow_{\sqcup}$ - rule doesn't apply to y_2:Y, Y $\sqsubseteq$ U $\sqcup$ V. We choose y_2:U since the constraint y:U is in S_2.

$\cup \{y_2R'z\}$ Now the $\rightarrow_{\exists}$ - rule doesn't apply to y_2:U, U($\exists$R')Z since the constraint y_2R'z is added and the constraint z:Z is in S_2.

The idea behind this "R_1-completion" is, that the individual variable y_2 has to be a copy of the individual variable y, which occurs in the R_2-complete constraint tree. It is easy to see that the constraint tree S_1 is R_1-complete. Furthermore the constraint tree S_1 doesn't contain constraints of the form x:A and x:$\neg$A. With proposition 4.2.1. we know that S_1 is consistent and hence the constraint tree S_2 is consistent.

Proposition 4.4.1 *Let S be an R_2-complete constraint tree not containing the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A. Then there exists an R_1-complete constraint tree S´ such that*

- *$S \subseteq S'$ and*
- *S´ doesn't contain the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A.*

Proof. First we define the set V_S of all tuples $(z,Z) \in V^I \times V^C$ for the constraint tree S, such that the $\rightarrow_{atleast}$ - rule of the R_1-completion rules applies to constraints of the form z:Z, Z(atleast N R):

$V_S = \{ (z,Z) \in V^I \times V^C \mid \{z:Z, Z(\text{atleast } N\ R)\} \subseteq S \text{ and } n_{R,S}(z) < N\}$.

Let S be an R_2-complete constraint tree not containing the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A. Consider the following two cases:

a) Suppose $V_S = \emptyset$. Then the constraint tree S is R_1-complete, because
   - the $\rightarrow_{atleast}$ - rule of the R_1-completion rules doesn't apply to S and
   - S is R_2-complete.

b) Now assume that $V_S \neq \emptyset$. Then we construct a constraint tree S´ from S, such that
   - S´ is R_2-complete,
   - $S \subseteq S'$,
   - S´ doesn't contain the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A and
   - $\| V_{S'} \| = \| V_S \| - 1$.

Thus after finitely many iterations – since V_S is finite for a finite constraint tree – we obtain an R_1-complete constraint tree not containing the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A.

Now let us construct a constraint tree S´ from S with the required properties: We choose $(x,X) \in V_S$, such that there is no pair $(y,Y) \in V_S$ where Y has a level greater

than X. Note that there is at least one element in V_S with this property. Then {x:X, X(atleast N R)} $\subseteq$ S with $n_{R,S}(x) < N$. In the following we often refer to the constraint xRy $\in$ S for a fixed individual variable y. Note that there is at least one constraint of the form xRy $\in$ S for some individual variable y because of the $\rightarrow_{atleast}$ - rule of the R_2-completion rules.

First we define the constraint tree

$S^1 = S \cup \{xRy_1,..., xRy_d \mid d = N - n_{R,S}(x), y_1, ..., y_d$ are new individual variables$\}$.

Observe that $\| V_S 1 \| = \| V_S \| - 1$ and that in general the constraint tree S^1 is not R_2-complete. In the following we add constraints to S^1 such that we obtain an R_2-complete constraint tree not containing constraints of the form x:A and x:$\neg$A.

Consider the constraint xRy $\in$ S. We add for every concept variable Y with y:Y the constraints y_1:Y, ..., y_d:Y and obtain the constraint tree

$S^2 = S^1 \cup \{y_1:Y, ..., y_d:Y \mid Y$ is a concept variable with $\{xRy, y:Y\} \subseteq S^1\}$.

We observe that the $\rightarrow_{\sqcup}$ - rule doesn't apply to S^2: Suppose $\{y:Y, Y \sqsubseteq U \sqcup V\} \subseteq S^1$. The constraint tree S^1 contains either y:U or y:V because S is R_2-complete. By definition the constraint tree S^2 contains either the constraints y_i:U or y_i:V for $1 \leq i \leq d$. Thus the $\rightarrow_{\sqcup}$ - rule doesn't apply to S^2.

Now we add for every concept symbol A with y:A the constraints y_1:A, ..., y_d:A which gives the constraint tree:

$S^3 = S^2 \cup \{y_1:A, ..., y_d:A \mid A$ is a concept symbol with $\{xRy, y:A\} \subseteq S^2\}$.

It is easy to see that the $\rightarrow_{\sqsubseteq}$ - rule doesn't apply to S^3. Furthermore we note that the constraint tree S^3 doesn't contain the constraints x:A and x:$\neg$A for any $x \in V^I$ and any concept symbol A. Suppose xRy, y:A $\in S^2$. Then the constraints y_1:A, ..., y_d:A are added to S^2, where y_1, ..., y_d are new individual variables not occurring in S. We know that y:$\neg$A $\notin S^2$ and therefore the constraints y_i:$\neg$A $\notin S^3$ for $1 \leq i \leq d$. If the constraints xRy, y:$\neg$A $\in S^2$, similar arguments apply.

Next we add constraints to S^3, obtaining the constraint tree S´, such that the $\rightarrow_{\forall}$ -

rule and the $\rightarrow_{\exists}$ - rule don't apply to S´. The constraint tree S´ is obtained from S^3 by adding the constraints $y_1R´z$, ..., $y_dR´z$ for every $yR´z \in S^3$ for some role symbol R´ and some individual variable z:

$S´ = S^3 \cup \{y_1R´z, ..., y_dR´z \mid R´$ is a role symbol and z is an individual variable with $\{xRy, yR´z\} \subseteq S^3\}$.

We observe that the $\rightarrow_{\exists}$ - rule doesn't apply to S´: Suppose $\{y:Y, Y(\exists R´)Z\} \subseteq S^3$. Then the constraints yR´z and z:Z are in the constraint tree S^3, since S is R_2-complete. Thus for the constraints $y_i:Y$, $Y(\exists R´)Z$ $(1 \leq i \leq d)$ the constraints $y_iR´z$ $(1 \leq i \leq d)$ and z:Z are in S´, and the $\rightarrow_{\exists}$ - rule doesn't apply to S´. Furthermore for the constraints $y_i:Y$, $y_iR´z$, $Y(\forall R´)Z$ $(1 \leq i \leq d)$ the constraint tree S´ contains the constraints z:Z. Thus the $\rightarrow_{\forall}$ - rule doesn't apply to S´.

Let us now summarize the properties of the constraint tree S´. Remember that we chose $(x,X) \in V_S$ such that the level of X was maximal. Furthermore, S contains constraints x:X and X(atleast N R) where $n_{R,S}(x) < N$.

1) $n_{R,S´}(x) = N$.

2) The $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\sqsubseteq}$ - rule, the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\forall}$ - rule don't apply to S´.

3) The $\rightarrow_{atleast}$ - rule of the R_2-completion rules doesn't apply to S´.

Suppose $y_i:Y \in S´$ $(1 \leq i \leq d)$ where y_i is a new individual variable not occurring in the constraint tree S. Then there exist constraints x:X, xRy and y:Y for some individual variable y in S. By the choice of $(x,X) \in V_S$, we know that in S there are no constraints y:Y, Y(atleast M R´) with $n_{R´,S}(y) < M$. Hence, there is no constraint Y(atleast M R´) in S´ with $n_{R´,S´}(y_i) < M$, which implies that $(y_i,Y) \notin V_{S´}$. Thus the $\rightarrow_{atleast}$ - rule of the R_2 - completion rules doesn't apply to S´.

4) The $\rightarrow_{atmost}$ - rule doesn't apply to S´.

We know that there are no constraints y:Y, Y(atmost M R´) in S with $n_{R´,S}(y) > M$ because S is R_2-complete. Hence, there are no constraints $y_i:Y$, Y(atmost M R´) in S´ with $n_{R´,S´}(y_i) > M$ for $1 \leq i \leq d$ and the $\rightarrow_{atmost}$ - rule doesn't apply to S´.

5) The $\rightarrow_{error}$ - rule doesn't apply to S´.

There are no constraints y:Y, Y(atleast N R^), y:Z, Z(atmost M R) in S with N > M and hence there are no constraints y_i:Y, Y(atleast N R^), y_i:Z, Z(atmost M R) in S´ with N > M for $1 \leq i \leq d$.

6) S´ doesn't contain constraints of the form x:A and x:¬A.

As mentioned above, S^3 doesn't contain constraints of the form x:A and x:¬A.

7) $\| V_{S'} \| = \| V_S \| - 1$.

Now we have shown that no R_2-completion rule applies to S´. Thus we have constructed from an R_2-complete constraint tree S not containing constraints of the form x:A and x:¬A an R_2-complete constraint system S´ not containing constraints of the form x:A and x:¬A. Furthermore, $\| V_{S'} \| = \| V_S \| - 1$, and after finitely many iterations we obtain an R_1-complete constraint system. □

Theorem 4.4.2. *An R_2-complete constraint tree is inconsistent if and only if it contains, for some $x \in V^I$ and some concept symbol A, the constraints x:A and x:¬A.*

Proof. "⇒": Let S be an R_2-complete constraint tree not containing the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A. One can construct an R_1-complete constraint tree S´ from S, such that {x:A and x:¬A} $\notin$ S´ for any $x \in V^I$ and any concept symbol A (see proposition 4.3.1.). Using proposition 4.2.1. we conclude that S´ is consistent and hence S is consistent.

"⇐": As in proposition 4.2.1. □

Theorem 4.4.3. *Let S be a fresh constraint system obtained from an $\mathcal{ALCN}$- concept description C. We modify the function Completion of chapter 4.3. such that the $\rightarrow_{atleast}$ - rule of the R_2-completion rules is used instead of the R_1-completion rules.*

a) If C is consistent, then there exists a computation using Completion, such that the call Completion(S,0) returns an R_2-complete constraint system not containing x:A and x:¬A for any individual variable x and any concept symbol A.

b) If C is not consistent, then every call Completion(S,0) returns an R_2-complete constraint system containing x:A and x:¬A for some individual variable x and some concept symbol A.

Proof. Using theorem 4.3.2. and 4.4.2. □

5 PSPACE - completeness of $\mathcal{ALCN}$

In this chapter we show that checking the consistency of $\mathcal{ALCN}$- concept descriptions is PSPACE-complete. The idea behind our algorithm is, that it is not necessary to keep the whole constraint system in memory. We define subsets - so-called $R_{\mathcal{ALCN}}$-traces - of R_2-complete constraint systems with the following property: If each of these $R_{\mathcal{ALCN}}$-traces is consistent, then the whole constraint system is consistent. While the size of an R_2-complete constraint system can be exponential, the size of $R_{\mathcal{ALCN}}$-traces is linear in the size of the initial concept description.

Let S be a constraint system, R a role symbol and x an individual variable occurring in S. We want to count the number of $\exists$R-edges issuing from nodes X with x:X. We therefore define

$\exists\text{-edge}_{R,S}(x) := \|\{\ Y \mid \{x:X, X(\exists R)Y\} \subseteq S\ \}\|$.

We also want to keep track of the maximal number of constraints xRy such that the $\rightarrow_{atmost}$-rule doesn't apply to $\{x:X, X(\text{atmost N R})\} \subseteq S$ for some concept variable X. If there is no "atmost - restriction", then the value ∞ is returned. We thus define

$$\text{atmost}_{R,S}(x) := \begin{cases} \min\{N \mid \{x:X, X(\text{atmost N R})\} \subseteq S \text{ for some X}\} & \\ & \text{if } \{x:X, X(\text{atmost N R})\} \subseteq S \text{ for some x} \\ \infty & \text{otherwise.} \end{cases}$$

Consider an R_2-complete constraint system S containing an individual variable x with $\exists\text{-edge}_{R,S}(x) > \text{atmost}_{R,S}(x)$. Then we have

$\{x:X_1,\ X_1(\exists R)Y,\ x:X_2,\ X_2(\exists R)Z,\ xRy,\ y:Y,\ y:Z\} \subseteq S$

for some concept variables X_1, X_2, Y, Z and some individual variable y, that is, the constraint xRy is "distributed" to at least two $\exists$R-edges from nodes X with x:X.

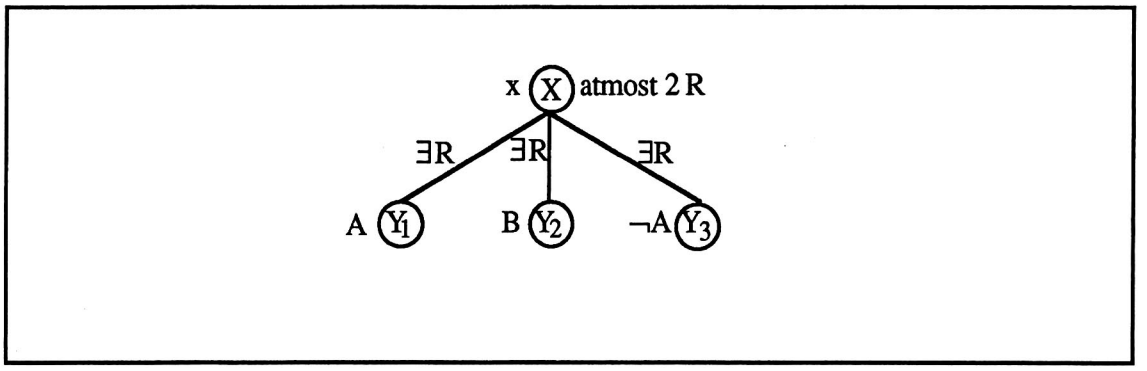

Figure 5.1. A constraint tree for the constraint system $S = \{x{:}X, X(\text{atmost } 2\ R), X(\exists R)Y_1, Y_1 \sqsubseteq A, X(\exists R)Y_2, Y_2 \sqsubseteq B, X(\exists R)Y_3, Y_3 \sqsubseteq \neg A\}$.

Example 5.1.

Consider the constraint system S given by the tree in figure 5.1. Since $\text{atmost}_{R,S}(x) = 2$, two constraints xRy, xRz are added to S, where y and z are new individual variables. Since $\exists\text{-edge}_{R,S}(x) = 3$, the following distributions are possible:

$S_1 = S \cup \{xRy, xRz\} \cup \{y{:}Y_1, y{:}Y_2, z{:}Y_3\} \cup \{y{:}A, y{:}B, z{:}\neg A\}$

$S_2 = S \cup \{xRy, xRz\} \cup \{y{:}Y_1, z{:}Y_2, y{:}Y_3\} \cup \{y{:}A, z{:}B, y{:}\neg A\}$

$S_3 = S \cup \{xRy, xRz\} \cup \{y{:}Y_1, z{:}Y_2, z{:}Y_3\} \cup \{y{:}A, z{:}B, z{:}\neg A\}$

Note that S_1, S_2 and S_3 are obtained from S by applications of the R_2-completion rules. Since at least one of the constraint systems S_1, S_2 and S_3, respectively, is consistent, we know that S is consistent.

For the following we need the following definitions:

Let $Y = \{Y_1, ..., Y_n\}$ be a set of concept variables and let m be a positive integer with $m < n$. We divide Y into m pairwise disjoint subsets $\pi_1, ..., \pi_m$, such that $\pi_1 \cup ... \cup \pi_m = Y$ and $\pi_i \neq \emptyset$ $(1 \leq i \leq m)$. The set $\{\pi_1, ..., \pi_m\}$ is called an **m-partition** of $\{Y_1, ..., Y_n\}$.

Given a constraint system S and individual variables x and y occurring in S, y is called a **successor** of x if S contains a constraint xRy.

Now we define completion rules, such that we obtain so-called $R_{\mathcal{ALCN}}$-traces for a fresh constraint system. The idea behind the $R_{\mathcal{ALCN}}$-traces is, that every individual variable occurring in an $R_{\mathcal{ALCN}}$-trace has at most one successor. We do this by restricting the R_2-completion rules.

The $R_{\mathcal{ALCN}}$-**trace rules** consist of the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\forall}$ - rule, the $\rightarrow_{\sqsubseteq}$ - rule, the $\rightarrow_{error}$ - rule and following three rules:

$S \rightarrow_{T\exists} \{y:Y, xRy\} \cup S$

if $x:X$ and $X(\exists R)Y$ are in S,

$\exists\text{-edge}_{R,S}(x) \leq \text{atmost}_{R,S}(x)$,

there is no constraint $xR'z$ in S, and y is a new individual variable.

$S \rightarrow_{Tatleast} \{xRy\} \cup S$

if $x:X$, $X(\text{atleast N R})$ are in S,

$\exists\text{-edge}_{R,S}(x) = 0$,

there is no constraint $xR'z$ in S, and y is a new individual variable.

$S \rightarrow_{Tatmost} S'$

if $\exists\text{-edge}_{R,S}(x) = n$, $\text{atmost}_{R,S}(x) = m$, $n > m$,

$\{x:X_1, X_1(\exists R)Y_1, ..., x:X_n, X_n(\exists R)Y_n\} \subseteq S$,

$\{\pi_1, ..., \pi_m\}$ is an m-partition of $\{Y_1, ..., Y_n\}$,

$S' \in \mathcal{S} = \{\{xRy_1\} \cup \{y_1:Z \mid Z \in \pi_1\} \cup S,$

$\{xRy_2\} \cup \{y_2:Z \mid Z \in \pi_2\} \cup S, ...,$

$\{xRy_m\} \cup \{y_m:Z \mid Z \in \pi_m\} \cup S\}\}$,

there is no constraint $xR'z$ in S, and $y_1, ..., y_m$ are new individual variables.

Let us now discuss the newly defined rules.

The $\rightarrow_{T\exists}$ - rule is a restriction of the $\rightarrow_{\exists}$ - rule, such that the $\rightarrow_{T\exists}$ - rule can be applied to at most one existential edge at every level. Hence every individual variable has

at most one successor.

The $\rightarrow_{T\text{atleast}}$ - rule forces the $\rightarrow_{\text{atleast}}$ - rule to be applied to S only, if $\exists\text{-edge}_{R,S}(x) = 0$ for some x. Otherwise, if $\exists\text{-edge}_{R,S}(x) > 0$, the $\rightarrow_{T\exists}$ - rule applies to S and there is no need to apply the $\rightarrow_{\text{atleast}}$ - rule.

Suppose $\exists\text{-edge}_{R,S}(x) > \text{atmost}_{R,S}(x) = m$ for some individual variable x occurring in S. Then the nondeterministic $\rightarrow_{T\text{atmost}}$ - rule applies, adds the single constraint xRy, and distributes it to at least one $\exists$R-edge. Consider the constraint system $U = \cup\{S' \mid S' \in \mathcal{S}\}$ where S´ ranges over all the possible constraint systems obtained with a fixed m-partition. Then $n_{R,U}(x) = \text{atmost}_{R,U}(x)$, and for every $\{x{:}X, X(\exists R)Y\} \subseteq U$ we have $\{xRy, y{:}Y\} \subseteq U$ for some y. Thus the $\rightarrow_{\text{atmost}}$ - rule doesn't apply to $\{x{:}X, X(\text{atmost N R})\} \subseteq U$ and the $\rightarrow_{\exists}$ - rule doesn't apply to $\{x{:}X, X(\exists R)Y\} \subseteq U$ for any concept variable X.

Let T be a constraint system obtained from a fresh constraint system S by application of $R_{\mathcal{ALCN}}$-trace rules. We call T an $R_{\mathcal{ALCN}}$-**trace** of S if no $R_{\mathcal{ALCN}}$-trace rule applies to T.

Proposition 5.1. *Let T be an $R_{\mathcal{ALCN}}$-trace. Then:*

a) If x occurs in T at level i, then every successor of x occurs at level i+1.

b) Every individual variable occurring in T has at most one successor.

c) At every level , there occurs at most one individual variable in T.

d) If T contains the constraints x:X and y:X, then x = y (that is, every concept variable has at most one individual variable associated with).

Proof.

a), b) follow by definition of the rules.

c) follows from a) and b), since there is only one individual variable at level 0.

d) follows, since every concept variable has a unique level. □

Propsition 5.2. *Let S be a fresh constraint system. The number of applications of $R_{\mathcal{ALCN}}$-trace rules to S is bounded linearly in the size of S.*

Proof. Follows from the fact that there is at most one individual variable x with x:X for a concept variable X in an $R_{\mathcal{ALCN}}$-trace of a fresh constraint system. □

Note that for a fresh constraint system there are finitely many $R_{\mathcal{ALCN}}$-traces modulo renaming of individual variables. In the following we show that we obtain an R_2-complete constraint system by the union of finitely many $R_{\mathcal{ALCN}}$-traces. Furthermore we will see that it is possible to compute all $R_{\mathcal{ALCN}}$-traces for a fresh constraint system such that at most one $R_{\mathcal{ALCN}}$-trace needs to be kept in memory. Since the size of $R_{\mathcal{ALCN}}$-traces is bounded linearly in the size of the fresh constraint system, polynomial space is needed. The next example demonstrates that it is not necessary to take the union of *all* $R_{\mathcal{ALCN}}$-traces to obtain an R_2-complete constraint system.

Example 5.2.

Consider the constraint system $S = \{x{:}X, X \sqsubseteq Y \sqcup Z, Y \sqsubseteq A, Z \sqsubseteq B\}$. Using the $R_{\mathcal{ALCN}}$-trace rules we obtain the following two $R_{\mathcal{ALCN}}$-traces:

1) $T_1 = S \cup \{x{:}Y, x{:}A\}$ and

2) $T_2 = S \cup \{x{:}Z, x{:}B\}$.

Since both T_1 and T_2 are R_2-complete constraint systems, it is sufficient to consider either T_1 or T_2. We will say that T_1 is an alternative trace of T_2, since only one of both $R_{\mathcal{ALCN}}$-traces is needed to obtain an R_2-complete constraint system.

Let T be an $R_{\mathcal{ALCN}}$-trace containing constraints x:X, $X \sqsubseteq Y \sqcup Z$, x:Y for some individual variable x and some concept variables X, Y and Z. Then an $R_{\mathcal{ALCN}}$-trace T′ is called an **alternative trace** of T, if T′ contains constraints x:X, $X \sqsubseteq Y \sqcup Z$, x:Z. Let S be a fresh constraint system and $\mathcal{T}_{\text{all}}$ be the set of the finitely many $R_{\mathcal{ALCN}}$-traces modulo

renaming of S. A subset $\mathcal{T}$ of $\mathcal{T}_{all}$ is called an $R_{\mathcal{ALCN}}$-**trace system** of S, if for every $T \in \mathcal{T}_{all}$ either T or an alternative trace of T is in $\mathcal{T}$.

Proposition 5.3. *Let $\mathcal{T}$ be an $R_{\mathcal{ALCN}}$-trace system. Then $S = \cup\{T \mid T \in \mathcal{T}\}$ is R_2-complete.*

Proof. Let $\mathcal{T}$ be an $R_{\mathcal{ALCN}}$-trace system. We will show that no R_2-completion rule applies to $S = \cup\{T \mid T \in \mathcal{T}\}$.

Consider the $\rightarrow_{\exists}$ - rule:

Assume that the $\rightarrow_{\exists}$ - rule applies to S. Then x:X, X(∃R)Y are in S and xRy, y:Y are not in S for any individual variable y.

1) If $\exists\text{-edge}_{R,S}(x) \leq \text{atmost}_{R,S}(x)$, then we obtain an $R_{\mathcal{ALCN}}$-trace $T \notin \mathcal{T}$ by applying the $\rightarrow_{T\exists}$ - rule to x:X, X(∃R)Y and we have a contradiction.

2) If $\exists\text{-edge}_{R,S}(x) > \text{atmost}_{R,S}(x)$, then there exists an $R_{\mathcal{ALCN}}$-trace $T \in \mathcal{T}$ containing x:X, X(∃R)Y, xRy, y:Y, because of the $\rightarrow_{\text{Tatmost}}$ - rule and we have a contradiction.

Thus our assumption is false and the $\rightarrow_{\exists}$ - rule doesn't apply to S.

Consider the $\rightarrow_{\text{atleast}}$ - rule:

Suppose $\{x{:}X, X(\text{atleast N R})\} \subseteq S$.

1) If $\exists\text{-edge}_{R,S}(x) > 0$, then $\{x{:}Y, Y(\exists R)Z, xRz, z{:}Z\} \subseteq S$ and $n_{R,S}(x) > 0$.

2) If $\exists\text{-edge}_{R,S}(x) = 0$, then the $\rightarrow_{\text{Tatleast}}$ - rule has applied to x:X, X(atleast N R) adding xRy. Hence $n_{R,S}(x) > 0$.

Thus the $\rightarrow_{\text{atleast}}$ - rule doesn't apply to S.

Consider the $\rightarrow_{\text{atmost}}$ - rule:

If $\{x{:}X, X(\text{atmost N R})\} \subseteq S$, then $n_{R,S}(x) \leq N$ because of the $\rightarrow_{\text{Tatmost}}$ - rule. Thus the $\rightarrow_{\text{atmost}}$ - rule doesn't apply to S.

It is easy to see that the other R_2-completion rules don't apply to S. Thus no R_2-completion rule applies to S and S is R_2-complete. □

Proposition 5.4. *A constraint system S is consistent if and only if there exists an $R_{\mathcal{ALCN}}$-trace system $\mathcal{T}$ of S, such that no $T \in \mathcal{T}$ contains the constraints x:A and x:¬A for any individual variable x and any concept symbol A.*

Proof. Let $\mathcal{T}$ be an $R_{\mathcal{ALCN}}$-trace system of S, such that no $R_{\mathcal{ALCN}}$-trace $T \in \mathcal{T}$ contains the constraints x:A and x:¬A for any individual variable x and any concept symbol A. Suppose x:A or x:¬A is in T for some concept symbol A. Then no $T' \in \mathcal{T}$ ($T' \neq T$) contains constraints of the form x:B or x:¬B for any concept symbol B. Since no $T \in \mathcal{T}$ contains x:A and x:¬A, we know that $S' = \cup\{T \mid T \in \mathcal{T}\}$ doesn't contain x:A and x:¬A. The constraint system S′ is $R_{\mathcal{ALCN}}$-complete (proposition 5.3.) and hence S is consistent.

Now suppose every $R_{\mathcal{ALCN}}$-trace system $\mathcal{T}$ of S contains an $R_{\mathcal{ALCN}}$-trace T, such that x:A and x:¬A are in T for some individual variable x and some concept symbol A. Then x:A and x:¬A are in S′ and S is inconsistent. □

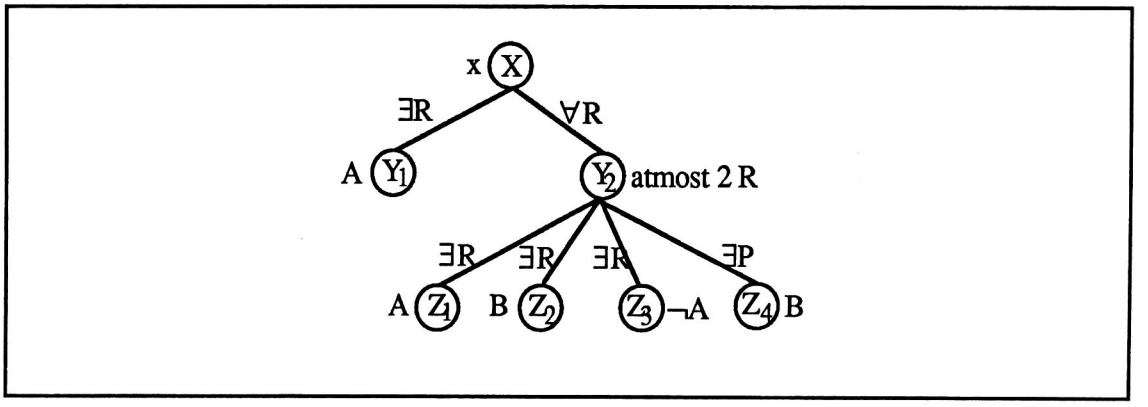

Figure 5.2. A constraint tree for the constraint system

S = {x:X, X(∃R)Y_1, $Y_1 \sqsubseteq$ A, X(∀R)Y_2, Y_2(atmost 2 R), Y_2(∃R)Z_1, $Z_1 \sqsubseteq$ A, Y_2(∃R)Z_2, $Z_2 \sqsubseteq$ B, Y_2(∃R)Z_3, $Z_3 \sqsubseteq$ ¬A, Y_2(∃P)Z_4, $Z_4 \sqsubseteq$ B}.

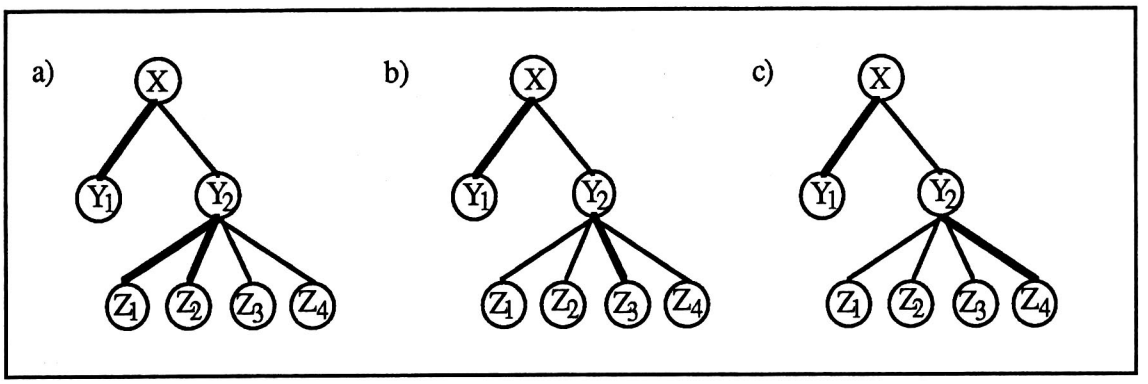

Figure 5.3. An $R_{\mathcal{ALCN}}$-trace system for the constraint system S. The ∃-edge from node X to node Y is drawn in bold, if x:X, X(∃R)Y, xRy, y:Y are in an $R_{\mathcal{ALCN}}$-trace.

Example 5.3.

Figure 5.2. shows a constraint tree for the constraint system

$$S = \{x{:}X,\ X(\exists R)Y_1,\ Y_1 \sqsubseteq A,\ X(\forall R)Y_2,\ Y_2(\text{atmost } 2\ R),\ Y_2(\exists R)Z_1,\ Z_1 \sqsubseteq A,$$
$$Y_2(\exists R)Z_2,\ Z_2 \sqsubseteq B,\ Y_2(\exists R)Z_3,\ Z_3 \sqsubseteq \neg A,\ Y_2(\exists P)Z_4,\ Z_4 \sqsubseteq B\}.$$

An $R_{\mathcal{ALCN}}$-trace system for S is shown in figure 5.3.

We now define the function *$\mathcal{ALCN}$-Completion* . The call *$\mathcal{ALCN}$-Completion (x,S)* computes up to renaming every $R_{\mathcal{ALCN}}$-trace of a fresh constraint system S with x as individual root. If there exists an $R_{\mathcal{ALCN}}$-trace system $\mathcal{T}$ of S, such that no $R_{\mathcal{ALCN}}$-trace $T \in \mathcal{T}$ contains the constraints x:A and x:¬A for any individual variable x and any concept symbol A, then the call returns *true* , otherwise *false*. Thus the function *$\mathcal{ALCN}$-Completion* yields *true* if and only if the constraint system S is consistent (see proposition 5.4.).

For the computation of $R_{\mathcal{ALCN}}$-traces it is redundant to keep constraints of the form xRy in memory, because x:X, X(∗R)Y, y:Y, where $* \in \{\forall, \exists\}$ implies the existence of a constraint xRy.

Function $\mathcal{ALCN}$-**Completion** (x,S) =

if ($\{x:A, x:\neg A\} \subseteq S$) $\vee$

($\{x:X, X(\text{atleast } N\ R), x:Y, Y(\text{atmost } M\ R)\} \subseteq S \ \wedge \ N > M$) $\vee$

($\{x:X, X(\text{atmost } O\ R), xRy\} \subseteq S$) (* 1 *)

then return *false*

else if $\{x:X, X \sqsubseteq A\} \subseteq S \ \wedge \ x:A \notin S$ (* 2 *)

then $\mathcal{ALCN}$-Completion (x, $\{x:A\} \cup S$)

else if $\{x:X, X \sqsubseteq Y \sqcup Z\} \subseteq S \ \wedge \ x:Y \notin S \ \wedge \ x:Z \notin S$ (* 3 *)

then $\mathcal{ALCN}$-Completion (x, $\{x:Y\} \cup S$) $\vee$ $\mathcal{ALCN}$-Completion (x, $\{x:Z\} \cup S$)

else if $\{x:X, X(\text{atleast } N\ R)\} \subseteq S \ \wedge \ \exists\text{-edge}_{R,S}(x) = 0$ (* 4 *)

then $\mathcal{ALCN}$-Completion (y, $\{y:U \mid \exists\{x:Z, Z(\forall R)U\} \subseteq S\} \cup S$)

else if [for all $\{x:X, X(\exists R)Y\} \subseteq S$ (* 5 *)

with $(\text{atmost}_{R,S}(x) = \infty) \ \vee \ (\text{atmost}_{R,S}(x) \geq \exists\text{-edge}_{R,S}(x))$ the call

$\mathcal{ALCN}$-Completion (y, $\{y:Y\} \cup \{y:U \mid \exists\{x:Z, Z(\forall R)U\} \subseteq S\} \cup S$),

where y is a new individual variable, returns *true*]

$\wedge$ [for all $\{x:X_1, X_1(\exists R)Y_1, ..., x:X_n, X(\exists R)Y_n\} \subseteq S$ (* 6 *)

with $(\text{atmost}_{R,S}(x) = m) \ \wedge \ (\exists\text{-edge}_{R,S}(x) = n) \ \wedge \ (m < n)$ the calls

$\mathcal{ALCN}$-Completion (y_1, $\{y_1:Z \mid Z \in \pi_1\} \cup \{y_1:V \mid \exists\{x:U, U(\forall R)V\} \subseteq S\} \cup S$)

$\wedge$ ... $\wedge$

$\mathcal{ALCN}$-Completion (y_m, $\{y_m:Z \mid Z \in \pi_m\} \cup \{y_m:V \mid \exists\{x:U, U(\forall R)V\} \subseteq S\} \cup S$),

where $\{\pi_1, ..., \pi_m\}$ is an m-partition of $\{Y_1, ..., Y_n\}$ and

$y_1, ..., y_m$ are new individual variables, return *true*

then return *true*]

else return *false*

end $\mathcal{ALCN}$-Completion.

Condition (* 1 *) tests if the $\rightarrow_{\text{error}}$ - rule applies to S. If so, then S is an inconsistent $R_{\mathcal{ALCN}}$-trace and *false* is returned.

If condition (* 2 *) is satisfied, then the $\rightarrow_{\sqsubseteq}$ - rule applies to S and we obtain a recursive call with x and $S \cup \{x:A\}$ as arguments.

If the $\rightarrow_{\sqcup}$ - rule applies to $\{x:X, X \sqsubseteq Y \sqcup Z\} \subseteq S$, then S is consistent if and only if either $S \cup \{x:Y\}$ or $S \cup \{x:Z\}$ is consistent. This is checked by condition (* 3 *) and we obtain two recursive calls.

Condition (* 4 *) tests if the $\rightarrow_{\text{Tatleast}}$ - rule applies to S. If so, then we add for every $\{x:Z, Z(\forall R)U\} \subseteq S$ the constraint y:U. Note that it is not necessary to add constraints of the form xRy. Thus, the constraint system given as argument to the recursive call is obtained from S by application of the $\rightarrow_{\text{Tatleast}}$ - rule and the $\rightarrow_{\forall}$ - rule and doesn't contain constraints of the form xRy.

If the $\rightarrow_{T\exists}$ - rule applies to S, then condition (* 5 *) is satisfied and for every $\{x:X, X(\exists R)Y\} \subseteq S$ with $(\text{atmost}_{R,S}(x) = \infty)$ or $(\text{atmost}_{R,S}(x) \geq \exists\text{-edge}_{R,S}(x))$, we have to check whether the recursive call

ALCN-Completion $(y, \{y:Y\} \cup \{y:U \mid \exists\{x:Z, Z(\forall R)U\} \subseteq S\} \cup S)$

returns *true* . We have to evaluate n further recursive calls, where $n = \exists\text{-edge}_{R,S}(x)$, which are put on a stack. Constraints of the form xRy are not added.

Condition (* 6 *) treats the case, that the $\rightarrow_{\text{Tatmost}}$ - rule applies to S. If $\text{atmost}_{R,S}(x) = m$ and $\exists\text{-edge}_{R,S}(x) = n$ and $m < n$ for some role symbol R, then we have to consider all m-partitions of $\{Y_1, ..., Y_n\}$. To ease our notation we assume that an m-partition of $\{Y_1, ..., Y_n\}$ is chosen nondeterministically. At this point the (linear space-) algorithm of proposition 5.5. is inserted and all m-partitions of $\{Y_1, ..., Y_n\}$ are enumerated. If there exists at least one m-partition of $\{Y_1, ..., Y_n\}$, such that we obtain an consistent $R_{\mathcal{ALCN}}$-trace system, then the call returns *true* .

Now let us apply the function *ALCN-Completion* to the constraint system of example 5.3. We obtain the call

ALCN-Completion (x, S) =

(* 1 *) *ALCN-Completion* $(y, \{y:Y_1, y:Y_2\} \cup S)$.

The $\rightarrow_{T\exists}$ - rule applies to x:X, X($\exists$R)Y_1 and the constraint y:Y_1 is added. Furthermore, because of x:X, X($\forall$R)Y_2, X($\exists$R)Y_1 the constraint y:Y_2 is added. Now the $\exists$R-edge and the $\forall$R-edge from node X are completed and the next recursive call (* 1 *) is evaluated.

ALCN-Completion (y, {y:Y_1, y:Y_2} $\cup$ S) =

(* 2 *) *ALCN-Completion* (y, {y:A} $\cup$ {y:Y_1, y:Y_2} $\cup$ S).

The constraints y:A is added, because of y:Y_1, $Y_1 \sqsubseteq A$, and we obtain the recursive call (* 2 *). Let S^1 = {y:A} $\cup$ {y:Y_1, y:Y_2} $\cup$ S). Then:

ALCN-Completion (y, S^1) =

(* 3 *) ([*ALCN-Completion* (z, {z:Z_1, z:Z_3} $\cup$ S^1) $\wedge$

(* 4 *) *ALCN-Completion* (u, {u:Z_2} $\cup$ S^1)] $\vee$

(* 5 *) [*ALCN-Completion* (z, {z:Z_1, z:Z_2} $\cup$ S^1) $\wedge$

(* 6 *) *ALCN-Completion* (u, {u:Z_3} $\cup$ S^1)] $\vee$

(* 7 *) [*ALCN-Completion* (z, {z:Z_1} $\cup$ S^1) $\wedge$

(* 8 *) *ALCN-Completion* (u, {u:Z_2, u:Z_3} $\cup$ S^1)]) $\wedge$

(* 9 *) *ALCN-Completion* (v, {v:Z_4} $\cup$ S^1)

Because $\exists$-edge$_{R,S^1}$(y) = 3 and atmost$_{R,S^1}$(y) = 2, the $\rightarrow_{Tatmost}$ - rule applies to S^1, and we have to consider all 2-partitions of {Z_1, Z_2, Z_3}. Further below we will see that there exists a linear space algorithm that enumerates all m-partitions of a finite set. In this example we assume that all 2-partitions of {Z_1, Z_2, Z_3} are simultaneously put on the stack. This is done by the call (* 3 *), ..., (* 8 *). If there exists at least one 2-partition (that is, a distribution of xRz, xRu to the three $\exists$R-edges), such that we obtain an R_2-complete constraint system not containing the constraints x:A and x:$\neg$A, then S in consistent. Furthermore, the $\rightarrow_{T\exists}$ - rule applies to S^1 and we obtain the call (* 9 *). Note that the recursive calls are put on a stack. Now the call (* 3 *) is taken from the stack and is evaluated. Then:

(* 3 *) *ALCN-Completion* (z, $\{z{:}Z_1, z{:}Z_3\} \cup S^1$) =

ALCN-Completion (z, $\{z{:}A\} \cup \{z{:}Z_1, z{:}Z_3\} \cup S^1$) =

ALCN-Completion (z, $\{z{:}\neg A\} \cup \{z{:}A, z{:}Z_1, z{:}Z_3\} \cup S^1$) = *false* .

Let $S^2 = \{z{:}Z_1, z{:}Z_3\} \cup S^1$. The $\rightarrow_{\sqsubseteq}$ - rule applies to S^2 and we obtain a constraint system containing z:A, z:¬A. Thus the call (* 3 *) is evaluated to *false* and there is no need to evaluate the call (* 4 *) . Now the call (* 5 *) is taken from the stack. Then:

(* 5 *) *ALCN-Completion* (z, $\{z{:}Z_1, z{:}Z_2\} \cup S^1$) =

ALCN-Completion (z, $\{z{:}A\} \cup \{z{:}Z_1, z{:}Z_2\} \cup S^1$) =

ALCN-Completion (z, $\{z{:}B\} \cup \{z{:}A, z{:}Z_1, z{:}Z_2\} \cup S^1$) = *true* .

Let $S^3 = \{z{:}B, z{:}A, z{:}Z_1, z{:}Z_2\} \cup S^1$. Then no $R_{\mathcal{ALCN}}$-trace rule applies to S^3. Since the constraint system S^3 doesn't contain constraints of the form x:A and x:¬A, the call returns *true*. Note that, if we add "enough" constraints of the form xRy to S^3, then we obtain the $R_{\mathcal{ALCN}}$-trace of S, that is shown in figure 5.3.a. Similarly the call (* 6 *) evaluates to *true*. Thus we have found a "consistent" distribution of xRz, xRu and need not to evaluate the calls (* 7 *) and (* 8 *) . Finally the call (* 9 *) evaluates to *true* . Thus the call *ALCN-Completion (x, S)* returns *true* and S is consistent.

If we stored all m-partitions of a finite set simultaneously on a stack, then we would need exponential space. Note that

$$\sum_{m=1}^{n} \| \text{m-partition of } \{1,2, ..., n\} \| \geq 2^{n-1} \ .$$

The algorithm NEXEQU [Nijenhuis/Wilf, 1975, page 81-85] enumerates all partitions (that is, all i-partitions for $i \geq 1$) of a finite set. Furthermore, the algorithm needs linear space in the size of the input.

Proposition 5.5. *Enumerating all i-partitions of a finite set can be done with linear space.*

Proof. By modifying the algorithm NEXEQU. □

Now let us consider the complexity of the function *ALCN-Completion* . The maximal recursion depth is the height of the given constraint tree. The function can be executed such that at most one $R_{\mathcal{ALCN}}$-trace needs to be kept in memory. Furthermore, for every level in an $R_{\mathcal{ALCN}}$-trace we have to administrate at most one partition which needs linear space in the size of the input constraint system. Hence the function *ALCN-Completion* needs at most quadratic space in the size of the input constraint system.

Theorem 5.6. *Checking consistency of ALCN- concept descriptions can be done with quadratic space.*

Proof. Let C be an *ALCN*- concept description. Then C is transformed in linear time into a fresh constraint system S. Let x be the individual root of S. The call *ALCN-Completion (x,S)* yields *true* if S is consistent and *false* otherwise. Furthermore, the function *ALCN-Completion* needs at most quadratic space in the size of S. □

Theorem 5.7. *Checking consistency of ALCN - concept descriptions is PSPACE-complete.*

Proof. The language *ALC* is a proper sublanguage of *ALCN*. Checking consistency of *ALC* - concept descriptions is PSPACE-complete [Schmidt-Schauß/Smolka 88]. With theorem 5.6. we conclude that checking consistency of *ALCN*- concept descriptions is PSPACE-complete. □

6 Role hierarchies

It was assumed by the definition of the languages $\mathcal{ALC}$ and $\mathcal{ALCN}$, that there is no further information about roles. We now define dependencies between roles and investigate their consequences for the consistency checking algorithm of the languages $\mathcal{ALC}$ and $\mathcal{ALCN}$. The idea is to model the *roleset differentiation* in KL-ONE: A roleset differentiates another when the former denotes a subrelation of the relation denoted by the latter [Brachman/Schmolze 1985].

One possibility to model the *roleset differentiation* is the introduction of the role-forming operator *androle* with the following semantics:

$I[(\text{androle } R_1, ..., R_n)] = \{ (a,b) \in \mathcal{D}^I \times \mathcal{D}^I \mid (a,b) \in I[R_1], ..., (a,b) \in I[R_n] \}$,

where $R_1, ..., R_n$ are role symbols. In BACK [Luck/Nebel/Peltason/Schmiedel 1987] the *androle*-operator is implemented in a restricted version: Only two arguments are permitted, and the second argument appears only in other androle expressions with the same first argument.

In this chapter we pursue another concept: A **role hierarchy** is a pair $(\mathcal{R}, \leq)$ such that $\mathcal{R}$ is a set of role symbols and $\leq$ is a partial order on $\mathcal{R}$. By abuse of notation we will also refer to the role hierarchy by the letter $\mathcal{R}$. Let $\mathcal{R}$ be a role hierarchy. An interpretation I is an **$\mathcal{R}$-interpretation** if $(a,b) \in I[P]$ implies $(a,b) \in I[R]$ for $P \leq R$ in $\mathcal{R}$. An $\mathcal{R}$-interpretation is an **$\mathcal{R}$-model** for a concept description C if $I[C]$ is nonempty. A concept description is **$\mathcal{R}$-consistent** if it has an $\mathcal{R}$-model. Furthermore we define **$\mathcal{R}$-subsumption** and **$\mathcal{R}$-equivalence** as one would expect.

The languages $\mathcal{ALC}$ and $\mathcal{ALCN}$ combined with the definition of role hierarchies are called $\mathcal{ALCR}$ and $\mathcal{ALCNR}$, respectively.

The androle concept and the role hierarchy concept are very similar. Differences occur since (androle R P) defines a new role, which is exactly the intersection of the roles P and R. On the other hand, if $Q \leq R$ and $Q \leq P$ then $I[Q]$ is a subset of the intersection of $I[R]$ and $I[P]$ for every $\mathcal{R}$-model I.

6.1. Simplification and unfolding of $\mathcal{ALCR}$ and $\mathcal{ALCNR}$ - concept descriptions

Let C be an $\mathcal{ALCR}$ or $\mathcal{ALCNR}$ - concept description. Using the simplification rules of chapter 2 we obtain a simple concept description C′. Before the concept description C′ is transformed into a constraint system we need the following definitions:

Let $\mathcal{R}$ be a role hierarchy. An I-assignment α is an $I_{\mathcal{R}}$-**assignment** if $(\alpha(x),\alpha(y)) \in I[P]$ implies $(\alpha(x),\alpha(y)) \in I[R]$ for $P \leq R$ in $\mathcal{R}$. $\mathbf{ASS}^{I}{}_{\mathcal{R}}$ is the set of all $I_{\mathcal{R}}$-assignments. An $\mathcal{R}$-interpretation I is an $\mathcal{R}$-**model** for a constraint system S if there exists an $I_{\mathcal{R}}$-assignment $\alpha \in \mathrm{ASS}^{I}{}_{\mathcal{R}}$ such that $\alpha \in I[S]$. A constraint system is $\mathcal{R}$-**consistent** if it has an $\mathcal{R}$-model.

The next proposition shows the relationship between simple $\mathcal{ALCR}(\mathcal{ALCNR})$ - concept descriptions and constraint systems.

Proposition 6.1.1. *Let x be an individual variable and X be a concept variable. A simple $\mathcal{ALCR}(\mathcal{ALCNR})$ - concept description C is $\mathcal{R}$-consistent if and only if the constraint system $\{x{:}X, X \sqsubseteq C\}$ is $\mathcal{R}$-consistent.*

Proof. Similar to the proof of proposition 3.1. □

Using the unfolding rules of chapter 3, the constraint system {x:X, X ⊑ C} is transformed into a simple constraint system S preserving $\mathcal{R}$-consistency and $\mathcal{R}$-inconsistency. For the completion of the simple constraint system S we need the following completion rule:

Let S be a constraint system. Then:

$S \rightarrow_{\leq} \{xRy\} \cup S$

if xPy is in S, $P \leq R$, and xRy is not in S.

The next proposition shows that the application of the $\rightarrow_{\leq}$ - rule preserves $\mathcal{R}$-consistency and $\mathcal{R}$-inconsistency.

Proposition 6.1.2. *Let S be a constraint system. If the constraint xPy is in S and $P \leq R$, then S is $\mathcal{R}$-consistent if and only if $S \cup \{xRy\}$ is $\mathcal{R}$-consistent.*

Proof. Let S be a constraint system containing xPy and let $P \leq R$.

If S is $\mathcal{R}$-consistent, then there exists an $I_{\mathcal{R}}$-assignment α with $(\alpha(x),\alpha(y)) \in I[P]$ and $(\alpha(x),\alpha(y)) \in I[P]$ implies $(\alpha(x),\alpha(y)) \in I[R]$. Hence $S \cup \{xRy\}$ is $\mathcal{R}$-consistent.

If S is not $\mathcal{R}$-consistent, then $I[S] = \emptyset$ and $I[S'] = I[S] \cap I[xRy] = \emptyset$. Thus S′ is not $\mathcal{R}$-consistent. □

6.2. Consistency checking of $\mathcal{ALCR}$ - concept descriptions

In the following we give rules to complete constraint systems obtained from $\mathcal{ALCR}$-concept descriptions.

The $\mathbf{R}_{\mathcal{ALCR}}$**-completion rules** consist of the $\rightarrow_\forall$ - rule, the $\rightarrow_\exists$ - rule, the $\rightarrow_\sqcup$ - rule, the $\rightarrow_\sqsubseteq$ - rule and the $\rightarrow_\leq$ - rule.

A constraint system is $\mathbf{R}_{\mathcal{ALCR}}$**-complete** if no $R_{\mathcal{ALCR}}$-completion rule applies to it.

Proposition 6.2.1. *Let S and S´be constraint systems. Then:*

a) If S´ is obtained from S by application of the (deterministic) $\rightarrow_\forall$ - rule, $\rightarrow_\exists$ - rule, $\rightarrow_\sqsubseteq$ - rule or $\rightarrow_\leq$ - rule, then S is $\mathcal{R}$-consistent if and only if S´ is $\mathcal{R}$-consistent.

b) If S´ is obtained from S by application of the (nondeterministic) $\rightarrow_\sqcup$ - rule, then S is $\mathcal{R}$-consistent if S´ is $\mathcal{R}$-consistent. Furthermore, there is a choice for S´ such that S´ is $\mathcal{R}$-consistent if and only if S is $\mathcal{R}$-consistent.

Proof. In proposition 4.1.1. it was shown, that the application of the $\rightarrow_\forall$ - rule, the $\rightarrow_\exists$ - rule, the $\rightarrow_\sqsubseteq$ - rule and the $\rightarrow_\sqcup$ - rule preserve consistency and inconsistency. Furthermore, these rules preserve $\mathcal{R}$-consistency and $\mathcal{R}$-inconsistency. In proposition 6.1.2. we proved, that the application of the $\rightarrow_\leq$ - rule preserves $\mathcal{R}$-consistency and $\mathcal{R}$-inconsistency. □

Proposition 6.2.2. *An $R_{\mathcal{ALCR}}$-complete constraint system is $\mathcal{R}$-inconsistent if and only if it contains for some $x \in V^I$ and some concept symbol A, the constraints x:A and x:¬A.*

Proof. "⇒": Let S be an $R_{\mathcal{ALCR}}$-complete constraint system not containing the constraints x:A and x:¬A for any $x \in V^I$ and any concept symbol A. Consider the standard interpretation I_s and the standard I-assignment α_s for the constraint system S. In proposition 4.2.1. it was shown that $\alpha_s \in I_s[S]$. Furthermore $(\alpha_s(x), \alpha_s(y)) \in I[P]$ implies $(\alpha_s(x), \alpha_s(y)) \in I[R]$ for $P \leq R$, because of the $\rightarrow_\leq$ - rule. Hence $\alpha_s \in ASS^I_{\mathcal{R}}$ and the constraint system S is $\mathcal{R}$-consistent.

"⇐": As in proposition 4.2.1. □

In the following we will show how to compute an $R_{\mathcal{ALCR}}$-complete constraint system for a fresh constraint system.

Theorem 6.2.3. *Let S be a constraint system obtained from an $\mathcal{ALCR}$ - concept description and let $\mathcal{R}$ be a role hierarchy. Then in at most exponentially many propagation steps depending on the size of S and $\mathcal{R}$ (where the size of $\mathcal{R}$ is the number of elements in $\mathcal{R}$) one can nondeterministically compute an $R_{\mathcal{ALCR}}$-complete constraint system S', such that S is $\mathcal{R}$-consistent if and only if S' is $\mathcal{R}$-consistent.*

Proof. Consider the function *Completion* from chapter 4.1. There we showed that the call *Completion(S,0)* computes nondeterministically an R_1-complete constraint system for a constraint system S given as argument. We modify this function in the following manner: Suppose *Completion(S,i)* is called. Let S_i be the set of all constraints at level i. Then the constraint system $S_{i\text{-complete}}$ is obtained from S_i by

$S_i \Rightarrow_{\sqcup} S_{\sqcup} \Rightarrow_{\exists} S_{\exists} \Rightarrow_{\leq} S_{\leq} \Rightarrow_{\forall} S_{\forall} \Rightarrow_{\sqsubseteq} S_{i\text{-complete}}$.

Now we will show that the constraint system $S_{i\text{-complete}}$ is $R_{\mathcal{ALCR}}$-complete at level i; that is, no $R_{\mathcal{ALCR}}$-completion rule applies to constraints at level i in $S_{i\text{-complete}}$:

Consider the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule:

Constraints of the form x:X, which have been introduced by the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\forall}$ - rule, are at level i+1. If the constraint x:X is at level i in $S_{i\text{-complete}}$, then x:X is already in $S_{\sqcup}$. Therefore the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule don't apply to $S_{i\text{-complete}}$.

Consider the $\rightarrow_{\leq}$ - rule:

Since the $\rightarrow_{\forall}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule, which are applied after the $\rightarrow_{\leq}$ - rule, don't introduce constraints of the form xRy, the $\rightarrow_{\leq}$ - rule doesn't apply to $S_{i\text{-complete}}$.

Consider the $\rightarrow_\forall$ - rule:

Since the $\rightarrow_\sqsubseteq$ - rule, which is applied after the $\rightarrow_\forall$ - rule, doesn't introduce constraints of the form x:X and xRy, the $\rightarrow_\forall$ - rule doesn't apply to $S_{i\text{-complete}}$.

Now we have shown that no $R_{\mathcal{ALCR}}$-completion rule applies to constraints at level i in $S_{i\text{-complete}}$, and hence $S_{i\text{-complete}}$ is complete at level i.

Since a constraint system S with $\|S\| = m$ doesn't contain constraints at level M with $M \geq m$, after finitely many iterations the modified function terminates and computes the $R_{\mathcal{ALCR}}$-complete constraint system S^m. Note that the $\rightarrow_\leq$ - rule applies at most $\|\mathcal{R}\| * \|S_\exists\|$ times. Thus at most $\|\mathcal{R}\| * \|S_\exists\|$ constraints are added. As in theorem 4.1.4. one can construct a polynomial p, such that $\|S^m\| \leq e^{(m+1)} * p(m)$, and we conclude that in at most exponentially many propagation steps one can nondeterministically compute an $R_{\mathcal{ALCR}}$-complete constraint system. □

6.3. PSPACE - completeness of $\mathcal{ALCR}$

In the following we will prove that checking consistency of $\mathcal{ALCR}$-concept descriptions is PSPACE-complete. The idea is that an $R_{\mathcal{ALCR}}$-complete constraint system can be obtained as the union of so-called $R_{\mathcal{ALCR}}$-traces. If none of these $R_{\mathcal{ALCR}}$-traces contains the constraints x:A and x:¬A, then the $R_{\mathcal{ALCR}}$-complete constraint system doesn't contain the constraints x:A and x:¬A. While the size of an $R_{\mathcal{ALCR}}$-complete constraint system can be exponential, the size of $R_{\mathcal{ALCR}}$-traces is linear in the size of the initial concept description and the role hierarchy.

First we define completion rules, such that we obtain so-called $R_{\mathcal{ALCR}}$-traces for a fresh constraint system. $R_{\mathcal{ALCR}}$-traces have the property that every individual variable occurring in an $R_{\mathcal{ALCN}}$-trace has at most one successor.

The $R_{\mathcal{ALCR}}$-**trace rules** consist of the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\sqsubseteq}$ - rule and the following two rules:

$S \rightarrow_{T\exists} \{y:Y, xRy\} \cup S$

if x:X and X(∃R)Y are in S, there is no constraint xR´z in S, and y is a new individual variable.

$S \rightarrow_{T\forall} \{y:Y\} \cup S$

if x:X, xPy, X(∀R)Y are in S, $P \leq R$, and y:Y is not in S.

Let T be a constraint system obtained from a fresh constraint system S by application of $R_{\mathcal{ALCR}}$trace rules. We call T an $R_{\mathcal{ALCR}}$**trace** of S, if no $R_{\mathcal{ALCR}}$trace rule applies to T.

Proposition 6.3.1. *Let T be an $R_{\mathcal{ALCR}}$-trace. Then:*

a) If x occurs at level i, then every successor of x occurs at level i+1.

b) Every individual variable occurring in T has at most one successor.

c) At every level, there occurs at most one individual variable in T.

d) If T contains the constraints x:X and y:X, then x = y (that is, every concept variable has at most one individual variable associated with).

Proof. As in proposition 5.1. □

Proposition 6.3.2. *Let S be a fresh constraint system and $\mathcal{R}$ be a role hierarchy. The number of applications of $R_{\mathcal{ALCR}}$-trace rules to S is bounded linearly in the size of S and $\mathcal{R}$.*

Proof. Follows from the fact that for a concept variable X there is at most one individual variable x with x:X in an $R_{\mathcal{ALCR}}$trace of a fresh constraint system. □

Let T be an $R_{\mathcal{ALCR}}$-trace containing constraints x:X, $X \sqsubseteq Y \sqcup Z$, x:Y for some individual variable x and some concept variables X, Y and Z. Then an $R_{\mathcal{ALCR}}$-trace T′ is called an **alternative trace** of T, if T′ contains constraints x:X, $X \sqsubseteq Y \sqcup Z$, x:Z. Let S be a fresh constraint system and $\mathcal{T}_{all}$ be the set of the finitely many $R_{\mathcal{ALCR}}$-traces modulo renaming of S. A subset $\mathcal{T}$ of $\mathcal{T}_{all}$ is called an $R_{\mathcal{ALCR}}$-**trace system** of S, if for every $T \in \mathcal{T}_{all}$ either T or an alternative trace of T is in $\mathcal{T}$.

Proposition 6.3.3. *Let $\mathcal{T}$ be an $R_{\mathcal{ALCR}}$-trace system and let $T_0 = \cup\{T \mid T \in \mathcal{T}\}$. Then the constraint system $S = T_0 \cup \{ xRy \mid xPy \in T_0, P \leq R\}$ is $R_{\mathcal{ALCR}}$-complete.*

Proof. Let $\mathcal{T}$ be an $R_{\mathcal{ALCR}}$-trace system and let $T_0 = \cup\{T \mid T \in \mathcal{T}\}$. We will show that no $R_{\mathcal{ALCR}}$-completion rule applies to $S = T_0 \cup \{ xRy \mid xPy \in T_0, P \leq R\}$.

Consider the $\rightarrow_{\exists}$ - rule:

Assume that the $\rightarrow_{\exists}$ - rule applies to S. Then x:X, X(∃R)Y are in S and xRy is not in S for any individual variable y. But then we obtain a trace $T \notin \mathcal{T}$ by applying the $\rightarrow_{T\exists}$ - rule to x:X, X(∃R)Y and we have a contradiction. Thus our assumption is false and the $\rightarrow_{\exists}$ - rule doesn't apply to S.

Consider the $\rightarrow_{\forall}$ - rule:

Assume that the $\rightarrow_{\forall}$ - rule applies to S. Then x:X, xRy, X(∀R)Y are in S and y:Y is not in S. Two cases are possible.

1) If xRy is in T_0, then x:Z, Z(∃R)U are in T_0. Let $T \in \mathcal{T}$ be an $R_{\mathcal{ALCR}}$-trace containing x:Z, Z(∃R)U, xRy. Note that such an $R_{\mathcal{ALCR}}$-trace exists. Then y:Y is in T, because of the $\rightarrow_{T\forall}$ - rule. Hence y:Y is in S and we have a contradiction.

2) If xRy is not in T_0, then x:Z, Z(∃P)U, xPu are in T_0 and $P \leq R$. Then there exists an $R_{\mathcal{ALCR}}$-trace $T \in \mathcal{T}$ containing x:Z, Z(∃R)U, xPu. But then $y:Y \in T$, because of the $\rightarrow_{T\forall}$ - rule. Hence $y:Y \in S$, and we have a contradiction.

Thus our assumption is false and the $\rightarrow_{\forall}$ - rule doesn't apply to S.

It is easy to see that the $\rightarrow_{\sqcup}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule don't apply to S. Furthermore,

the $\rightarrow_{\leq}$ - rule doesn't apply to S because of construction of the constraint system S. Thus no $R_{\mathcal{ALCR}}$-completion rule applies to S and S is $R_{\mathcal{ALCR}}$-complete. □

Proposition 6.3.4. *A constraint system S is $\mathcal{R}$-consistent if and only if there exists an $R_{\mathcal{ALCR}}$-trace system $\mathcal{T}$ of S, such that no $T \in \mathcal{T}$ contains the constraints x:A and x:¬A for any individual variable x and any concept symbol A.*

Proof. Let $\mathcal{T}$ be an $R_{\mathcal{ALCR}}$-trace system of S, such that no $R_{\mathcal{ALCR}}$-trace $T \in \mathcal{T}$ contains the constraints x:A and x:¬A for any individual variable x and any concept symbol A. Suppose x:A or x:¬A is in T for some concept symbol A. Then no $T' \in \mathcal{T}$ ($T' \neq T$) contains constraints of the form x:B or x:¬B for any concept symbol B. Since no $T \in \mathcal{T}$ contains x:A and x:¬A, we know that $T_0 = \cup\{T \mid T \in \mathcal{T}\}$ doesn't contain x:A and x:¬A. Then $T_0 \cup \{ xRy \mid xPy \in T_0, P \leq R\}$ is $R_{\mathcal{ALCR}}$-complete (proposition 6.3.3.) and hence S is $\mathcal{R}$-consistent.

Now suppose every $R_{\mathcal{ALCR}}$-trace system $\mathcal{T}$ of S contains an $R_{\mathcal{ALCR}}$-trace T, such that x:A and x:¬A are in T for some individual variable x and some concept symbol A. Then x:A and x:¬A are in T_0 and S is $\mathcal{R}$-inconsistent. □

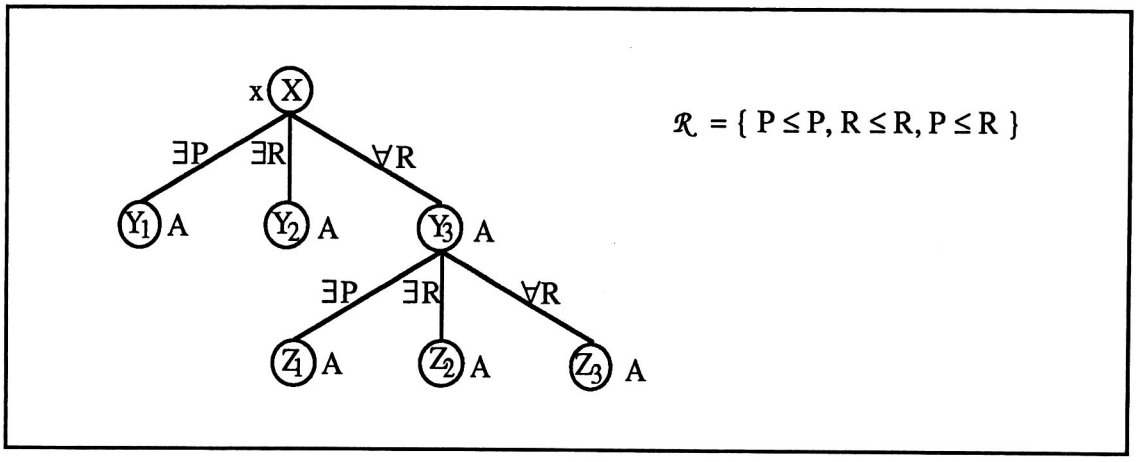

Figure 6.3.1. A constraint tree for the fresh constraint system

S = {x:X, $X(\exists P)Y_1$, $Y_1 \sqsubseteq A$, $X(\exists R)Y_2$, $Y_2 \sqsubseteq A$, $X(\forall R)Y_3$, $Y_3(\exists P)Z_1$, $Z_1 \sqsubseteq A$, $Y_3(\exists R)Z_2$, $Z_2 \sqsubseteq A$, $Y_3(\forall R)Z_3$, $Z_3 \sqsubseteq A$} with $\mathcal{R} = \{P \leq P, R \leq R, P \leq R\}$.

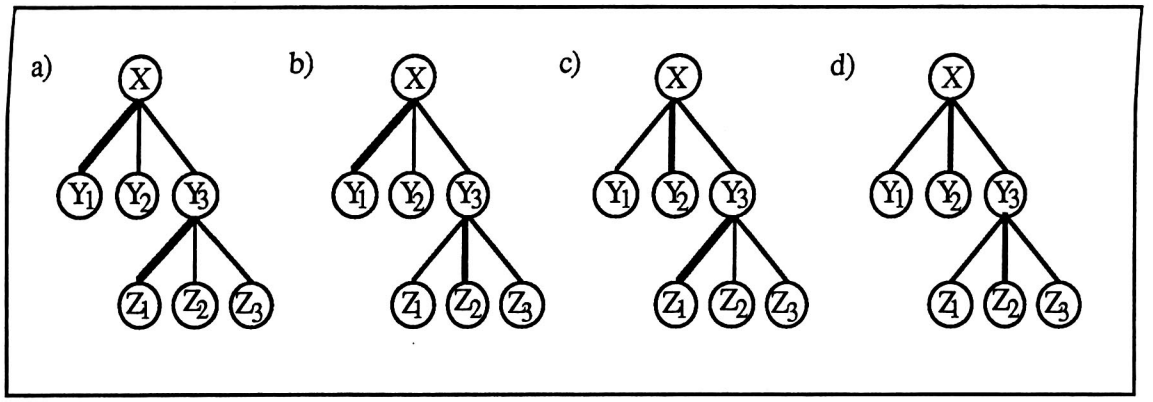

Figure 6.3.2. $R_{\mathcal{ALCR}}$-traces for the constraint system S. The ∃-edge from node X to node Y is drawn in bold, if the $\rightarrow_{T\exists}$ - rule has applied to x:X, X(∃R)Y.

Example 6.3.1.

Consider the constraint tree in figure 6.3.1. The $R_{\mathcal{ALCR}}$-traces for this constraint tree are shown in figure 6.3.2.

We now define the function *ALCR-Completion* . The call *ALCR-Completion(x,S)* computes up to renaming every $R_{\mathcal{ALCR}}$-trace of a fresh constraint system S with x as individual root. If there exists an $R_{\mathcal{ALCR}}$-trace system $\mathcal{T}$ of S, such that no $R_{\mathcal{ALCR}}$-trace $T \in \mathcal{T}$ contains the constraints x:A and x:¬A for any individual variable x and any concept symbol A, then the call returns *true*, otherwise *false*. Thus the function *ALCR-Completion* yields *true* if and only if the constraint system S is consistent (see proposition 6.3.4.).

As mentioned before, it is not necessary to keep constraints of the form xRy in memory.

Function $\mathcal{ALCR}$-**Completion** (x,S) =

if $\{x{:}A, x{:}\neg A\} \subseteq S$ then return *false* (* 1 *)

else if $\{x{:}X, X \sqsubseteq A\} \subseteq S \ \wedge \ x{:}A \notin S$ (* 2 *)

then $\mathcal{ALCR}$-Completion (x, $\{x{:}A\} \cup S$)

else if $\{x{:}X, X \sqsubseteq Y \sqcup Z\} \subseteq S \ \wedge \ x{:}Y \notin S \ \wedge \ x{:}Z \notin S$ (* 3 *)

then $\mathcal{ALCR}$-Completion (x, $\{x{:}Y\} \cup S$) $\vee$ $\mathcal{ALCR}$-Completion (x, $\{x{:}Z\} \cup S$)

else if for all $\{x{:}X, X(\exists P)Y\} \subseteq S$ the call (* 4 *)

$\mathcal{ALCR}$-Completion (y, $\{y{:}Y\} \cup \{y{:}U \mid \exists\{x{:}Z, Z(\forall R)U\} \subseteq S, P \leq R\} \cup S$),

where y is a new individual variable, returns *true*

then return *true*

else return *false*

end $\mathcal{ALCR}$-Completion.

Condition (* 1 *) tests if S is an $\mathcal{R}$-inconsistent constraint system.

If condition (* 2 *) is satisfied, then the $\rightarrow_{\sqsubseteq}$ - rule applies to S and we obtain a recursive call with x and $S \cup \{x{:}A\}$ as arguments.

If the $\rightarrow_{\sqcup}$ - rule applies to $\{x{:}X, X \sqsubseteq Y \sqcup Z\} \subseteq S$, then S is $\mathcal{R}$-consistent if and only if either $S \cup \{x{:}Y\}$ or $S \cup \{x{:}Z\}$ is $\mathcal{R}$-consistent. This is checked by condition (* 3 *) and we obtain two recursive calls.

If the $\rightarrow_{T\exists}$ - rule applies to S, then condition (* 4 *) is satisfied and for every $\{x{:}X, X(\exists P)Y\} \subseteq S$, we have to check whether the recursive call

$\mathcal{ALCR}$-Completion (y, $\{y{:}Y\} \cup \{y{:}U \mid \exists\{x{:}Z, Z(\forall R)U\} \subseteq S, P \leq R\} \cup S$)

returns *true*. Note that it is not necessary to add constraints of the form xRy. Thus the constraint system given as argument to the recursive call is obtained from S by application of the $\rightarrow_{T\exists}$ - rule, the $\rightarrow_{\forall}$ - rule and the $\rightarrow_{\leq}$ - rule. We have to evaluate n further recursive calls, where $n = \exists\text{-edge}_{R,S}(x)$, which are put on a stack.

Now let us apply the function *ALCR-Completion* to the constraint system of example 6.3.1. We obtain the following evaluations:

ALCR-Completion (x, S) =

(*1*) *ALCR-Completion* (y_1, $\{y_1{:}Y_1, y_1{:}Y_3\} \cup S$) $\wedge$

(*2*) *ALCR-Completion* (y_2, $\{y_2{:}Y_2, y_2{:}Y_3\} \cup S$).

Since the $\rightarrow_{T\exists}$ - rule applies to $x{:}X$, $X(\exists P)Y_1$ and $x{:}X$, $X(\exists R)Y_2$, the recursive calls (*1*) and (*2*) are put on a stack. Consider the call (*1*) : The constraint $y_1{:}Y_1$ is added to S because $\{x{:}X, X(\exists P)Y_1\} \subseteq S$, and $y_1{:}Y_3$ is added to S because $\{x{:}X, X(\exists P)Y_1, X(\forall R)Y_3\} \subseteq S$ and $P \leq R$. Now the call (*1*) is taken from the stack and is evaluated. Let $S^1 = \{y_1{:}Y_1, y_1{:}Y_3\} \cup S$. Then:

ALCR-Completion (y_1, S^1) =

(*3*) *ALCR-Completion* (y_1, $\{y_1{:}A\} \cup S^1$) =

(*4*) *ALCR-Completion* (z_1, $\{z_1{:}Z_1, z_1{:}Z_3\} \cup \{y_1{:}A\} \cup S^1$) $\wedge$

(*5*) *ALCR-Completion* (z_2, $\{z_2{:}Z_2, z_2{:}Z_3\} \cup \{y_1{:}A\} \cup S^1$).

Since $\{y_1{:}Y_1, Y_1 \sqsubseteq A\} \subseteq S^1$, $y_1{:}A$ is added and we obtain the call (*3*). Because there are two $\exists$-edges from node Y_3, the recursive calls (*4*) and (*5*) are put on a stack. Now (*4*) is taken from the stack. Let $S^2 = \{z_1{:}Z_1, z_1{:}Z_3\} \cup \{y_1{:}A\} \cup S^1$. Then:

ALCR-Completion (z_1, S^2) =

ALCR-Completion (z_1, $\{z_1{:}A\} \cup S^2$) = *true* .

First the $\rightarrow_{\sqsubseteq}$ - rule is applied to S^2 adding $z_1{:}A$. Let $S^3 = \{z_1{:}A\} \cup S^2$. Then no $R_{\mathcal{ALCR}}$-trace rule applies to S^3. Since S^3 doesn't contain the constraints $x{:}A$ and $x{:}\neg A$ for any individual variable x and any concept symbol A, *true* is returned. Note, if we add "enough" constraints of the form xRy to S^3, then we obtain an $R_{\mathcal{ALCR}}$-trace of S. Then (*5*) is taken from the stack and evualutes to *true*. Thus the call (*1*) returns *true*. Similarly the call (*2*) returns *true*, and the call *ALCR-Completion* (x, S) evaluates to *true*. Thus the constraint system S is consistent.

Now we consider the complexity of the function *$\mathcal{ALCR}$-Completion* . The maximal recursion depth is the height of the given constraint system. The function *$\mathcal{ALCR}$-Completion* can be executed such that, besides some control information, at most one $R_{\mathcal{ALCR}}$-trace needs to be kept in memory. Hence the function *$\mathcal{ALCR}$-Completion* needs at most linear space in the size of the input constraint system and the role hierarchy.

Theorem 6.3.5. *Checking $\mathcal{R}$-consistency of $\mathcal{ALCR}$-concept descriptions can be done with linear space.*

Proof. Let C be an $\mathcal{ALCR}$-concept description. Then C transformed in linear time into a fresh constraint system S. Let x be the individual root of S. The call *$\mathcal{ALCR}$-Completion (x, S)* yields *true* if S is $\mathcal{R}$-consistent and *false* otherwise. Furthermore, the function *$\mathcal{ALCR}$-Completion* needs at most linear space in the size of S and $\mathcal{R}$. □

Theorem 6.3.6. *Checking $\mathcal{R}$-consistency of $\mathcal{ALCR}$-concept descriptions is PSPACE-complete.*

Proof. Since $\mathcal{ALC}$ is a proper sublanguage of $\mathcal{ALCR}$ and the fact, that checking consistency of $\mathcal{ALC}$-concept descriptions is PSPACE-complete [Schmidt-Schauß/Smolka 88] we know that checking $\mathcal{R}$-consistency of $\mathcal{ALCR}$-concept descriptions is PSPACE-hard. With theorem 6.3.5. we conclude that checking $\mathcal{R}$-consistency of $\mathcal{ALCR}$-concept descriptions is PSPACE-complete. □

6.4. Consistency checking of $\mathcal{ALCNR}$ - concept descriptions

In the first part of this chapter we defined the language $\mathcal{ALCR}$, that is the language $\mathcal{ALC}$ amalgamated with role hierarchies. In 6.3. we have shown that checking the consistency

of $\mathcal{ALCR}$- concept descritions is PSPACE-complete. We now investigate the language $\mathcal{ALCNR}$, that is the language $\mathcal{ALC}$ supplemented by number restrictions and role hierarchies. We give rules to complete constraint systems obtained from $\mathcal{ALCNR}$- concept descriptions. Since an algorithm applying these rules needs exponential space, we leave the reader at this point with an open problem: does there exist a polynomial space algorithm for checking consistency of $\mathcal{ALCNR}$- concept descriptions?

The problem of combining number restrictions and role hierarchies is quite interesting. Nebel [88] shows that checking subsumption in $\mathcal{FL}^-$ extended either by number restrictions or by the androle operator (an operator to create new roles by conjoining them - comparable to role hierarchies) can be done in polynomial time, whereas $\mathcal{FL}^-$ extended simultaneously by number restrictions and the androle operator is NP-hard in the strong sense, that is, it is independent in which way the numbers are coded.

We obtain the **$R_{\mathcal{ALCNR}}$-completion rules** from the $R_{\mathcal{ALCR}}$-completion rules by adding following three rules. Let S be a constraint system. Then:

$S \rightarrow_{atleast} \{xRy\} \cup S$

if x:X, X(atleast N R) are in S, $n_{R,S}(x) < N$, and y is a new individual variable not occurring in S

$S \rightarrow_{atmost} \{y \leftarrow z\}\ S$

if x:X, X(atmost N R), xRy, xRz are in S with $n_{R,S}(x) > N$, and

whenever x:Y, Y(atleast M P), xPy, xPz are in S, $P \leq R$, then $n_{P,S}(x) > M$.

$S \rightarrow_{error} \{x:A, x:\neg A\}$

if x:X, X(atmost 0 R), xRy are in S or

if x:X, X(atleast N P), x:Y, Y(atmost M R) are in S, $N > M$, and $P \leq R$.

A constraint system is $\mathbf{R}_{\mathcal{ALCNR}}$**-complete** if no $R_{\mathcal{ALCNR}}$-completion rule applies to it.

The $\rightarrow_{\text{atleast}}$ - rule and the $\rightarrow_{\text{error}}$ - rule are obvious. The following example illustrates the $\rightarrow_{\text{atmost}}$ - rule:

Example 6.4.1.

Consider the constraint system

$S = \{x{:}X, X(\text{atmost } 2\ R), X(\text{atleast } 2\ P), X(\text{atleast } 1\ Q), X(\forall P)Y, Y \sqsubseteq A, X(\forall Q)Z,$

$Z \sqsubseteq \neg A\}$ and the role hierarchy $\mathcal{R} = \{ P \leq P, Q \leq Q, R \leq R, P \leq R, Q \leq R \}$.

A constraint tree for S is shown in figure 6.4.1. We obtain the constraint system S´ by applications of the $\rightarrow_{\text{atleast}}$ - rule and the $\rightarrow_{\leq}$ - rule:

$S' = S \cup \{xPy_1, xPy_2, xQz\} \cup \{xRy_1, xRy_2, xRz\}$

If we use either the substitution $\{y_1 \leftarrow y_2\}S'$ or $\{y_2 \leftarrow y_1\}S'$, then we obtain following infinite chain of completion steps:

$S' = S \cup \{xPy_1, xPy_2, xQz\} \cup \{xRy_1, xRy_2, xRz\}$

$\rightarrow_{\text{atmost}} \{y_2 \leftarrow y_1\}S' = S \cup \{xPy_1, xQz\} \cup \{xRy_1, xRz\}$

$\rightarrow_{\text{atleast}} S \cup \{xPy_1, xPy_2, xQz\} \cup \{xRy_1, xRz\}$

$\rightarrow_{\leq} \quad S \cup \{xPy_1, xPy_2, xQz\} \cup \{xRy_1, xRy_2, xRz\}$

...

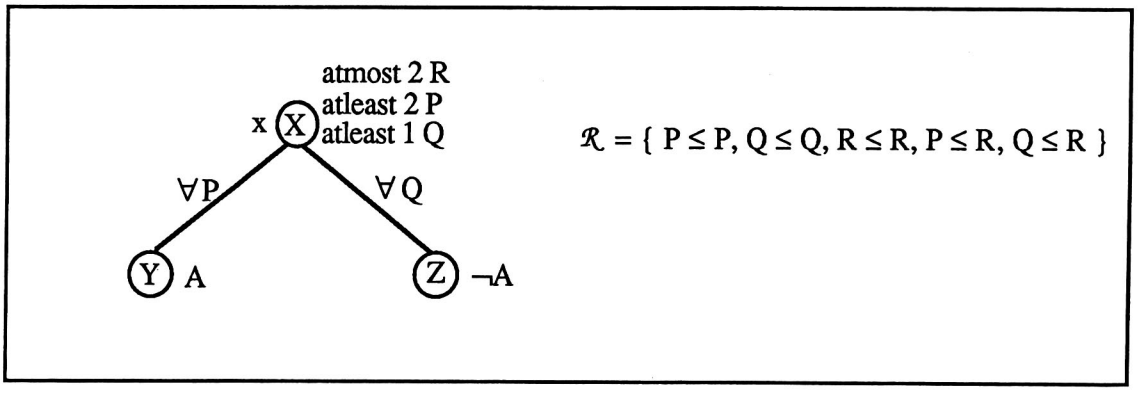

Figure 6.4.1. A constraint tree for the constraint system S.

Otherwise with the $\rightarrow_{\text{atmost}}$ - rule of the $R_{\mathcal{ALCNR}}$-completion rules we get a substitution $\{y_2 \leftarrow z\}$ and obtain

$$S' = S \cup \{xPy_1, xPy_2, xQz\} \cup \{xRy_1, xRy_2, xRz\}$$

$$\rightarrow_{\text{atmost}} \{y_2 \leftarrow z\}S' = S \cup \{xPy_1, xPz, xQz\} \cup \{xRy_1, xRz\},$$

and the $\rightarrow_{\text{atleast}}$ - rule does't apply.

Proposition 6.4.1. *Let S and S´ be constraint systems. Then:*

a) If S´ is obtained from S by application of the (deterministic) $\rightarrow_{\forall}$ - rule, $\rightarrow_{\exists}$ - rule, $\rightarrow_{\sqsubseteq}$ - rule, $\rightarrow_{\text{atleast}}$ - rule, $\rightarrow_{\text{error}}$ - rule or $\rightarrow_{\leq}$ - rule, then S is $\mathcal{R}$-consistent if and only if S´ is $\mathcal{R}$-consistent.

b) If S´ is obtained from S by application of the (nondeterministic) $\rightarrow_{\sqcup}$ - rule or $\rightarrow_{\text{atmost}}$ - rule, then S is $\mathcal{R}$-consistent if S´ is $\mathcal{R}$-consistent. Furthermore there is a choice for S such that S´ is $\mathcal{R}$-consistent if and only if S is $\mathcal{R}$-consistent.

Proof. Follows from propositions 4.1.1. and 6.1.2. □

Proposition 6.4.2. *An $R_{\mathcal{ALCNR}}$-complete constraint system is $\mathcal{R}$-inconsistent if and only if it contains for some $x \in V^I$ and some concept symbol A, the constraints x:A and $x:\neg A$.*

Proof. The proof is similar to the proofs in proposition 4.2.1. and 6.2.2. □

Theorem 6.4.3. *Let S be a constraint system obtained from an $\mathcal{ALCNR}$ - concept description. Then in at most exponentially many propagation steps depending on the size of S and $\mathcal{R}$ one can nondeterministically compute an $R_{\mathcal{ALCNR}}$-complete constraint system S´, such that S is $\mathcal{R}$-consistent if and only if S´ is $\mathcal{R}$-consistent.*

Proof. Consider the function *Completion* from chapter 4.1. There we showed that the call *Completion(S,0)* computes nondeterministically an R_1-complete constraint system for the constraint system given as argument. We modify this function in the following manner: Suppose *Completion(S,i)* is called. Let S_i be the set of all constraints at level i. The constraint system $S_{i\text{-complete}}$ is obtained from S_i by

$S_i \Rightarrow_{\sqcup} S_{\sqcup} \Rightarrow_{\exists} S_{\exists} \Rightarrow_{atleast} S_{atleast} \Rightarrow_{\leq} S_{\leq} \Rightarrow_{\forall} S_{\forall} \Rightarrow_{\sqsubseteq} S_{\sqsubseteq} \Rightarrow_{error} S_{error} \Rightarrow_{atmost} S_{i\text{-complete}}$.

Now we will show that the constraint system $S_{i\text{-complete}}$ is $R_{\mathcal{ALCNR}}$-complete at level i; that is, no $R_{\mathcal{ALCNR}}$-completion rule applies to constraints at level i in $S_{i\text{-complete}}$:

Consider the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule:

Constraints of the form x:X, which have been introduced by the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\forall}$ - rule, are at level i+1. If the constraint x:X is at level i in $S_{i\text{-complete}}$, then x:X is already in $S_{\sqcup}$. Therefore the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\exists}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule don't apply to $S_{i\text{-complete}}$.

Consider the $\rightarrow_{\leq}$ - rule:

Since the $\rightarrow_{\forall}$ - rule, the $\rightarrow_{\sqsubseteq}$ - rule, the $\rightarrow_{error}$ - rule, and the $\rightarrow_{atmost}$ - rule, which are applied after the $\rightarrow_{\leq}$ - rule, don't introduce constraints of the form xRy, the$\rightarrow_{\leq}$ - rule doesn't apply to $S_{i\text{-complete}}$.

Consider the $\rightarrow_{\forall}$ - rule:

Since the $\rightarrow_{\sqsubseteq}$ - rule, the $\rightarrow_{error}$ - rule and the $\rightarrow_{atmost}$ - rule, which are applied after the $\rightarrow_{\forall}$ - rule, don't introduce constraints of the form x:X and xRy, the $\rightarrow_{\forall}$ - rule doesn't apply to $S_{i\text{-complete}}$.

Consider the $\rightarrow_{atleast}$ - rule:

Assume that the $\rightarrow_{atleast}$ - rule applies to $S_{i\text{-complete}}$. Then $\{x:X,\ X(\text{atleast } N\ R)\} \subseteq S_{i\text{-complete}}$ and $N > \|\{\ y \mid xRy \in S_{i\text{-complete}}\}\|$. Since $N = \|\{\ y \mid xRy \in S_{atleast}\}\|$ after application of the $\rightarrow_{atleast}$ - rule, either

a) $\{x:Y,\ Y(\text{atmost } M\ R)\} \subseteq S_{i\text{-complete}}$ with $N > M$,

that is, only the first part of the $\rightarrow_{atmost}$ - rule has been satisfied or

b) $\{x{:}Z, Z(\text{atmost } M\ P), xPy, xPz\} \subseteq S_{\sqsubseteq}$ with $n_{P,S}(x) > M$ and

$\{x{:}X, X(\text{atleast } N\ R), xRy, xRz\} \subseteq S_{\sqsubseteq}$ with $n_{R,S}(x) = N$ and $R \leq P$,

that is, both parts of the $\rightarrow_{\text{atmost}}$ - rule have been satisfied.

Note that in the first case the $\rightarrow_{\text{error}}$ - rule would have applied to $S_{\sqsubseteq}$ and thus we have a contradiction. In the second case we know that the $\rightarrow_{\text{atmost}}$ - rule has applied to $\{x{:}Z, Z(\text{atmost } M\ P), xPy, xPz\} \subseteq S_{\sqsubseteq}$ although the condition for applying the $\rightarrow_{\text{atmost}}$ - rule to $S_{\sqsubseteq}$, has not been satisfied. Thus our assumption is false and the $\rightarrow_{\text{atleast}}$ - rule doesn't apply to $S_{\text{i-complete}}$.

Now we have shown that no $R_{\mathcal{ALCNR}}$-completion rule applies to constraints at level i in $S_{\text{i-complete}}$, and hence $S_{\text{i-complete}}$ is complete at level i.

Note that a constraint system S with $\|S\| = m$ doesn't contain constraints at level M with $M \geq m$. Thus after finitely many iterations the modified function terminates and computes the $R_{\mathcal{ALCNR}}$-complete constraint system S^m. Combining the estimates in the proofs of theorems 4.3.3. and 6.2.3. one can construct a polynomial p, such that $\|S^m\| \leq e^{(m+1)} * p(m)$, and we conclude that in at most exponentially many propagation steps one can nondeterministically compute an $R_{\mathcal{ALCNR}}$-complete constraint system. □

Why is it difficult to find a polynomial space algorithm for the problem of checking $\mathcal{R}$-consistency of $\mathcal{ALCNR}$ - concept descriptions? In chapter 5 we proved that checking consistency of $\mathcal{ALCN}$- concept descriptions is in PSPACE. There we used the following fact: Let S be a constraint system containing the constraints x:X and X(atleast N R). We showed, that it is not necessary to introduce N new constraints of the form xRy_1, ..., xRy_N, but it is sufficient to add the single constraint xRy. The following example demonstrates that this "trick" doesn't work here:

Example 6.4.2.

Consider the constraint system

$S = \{x{:}X, X(\text{atmost } 2\ R), X(\text{atleast } 2\ P), X(\text{atleast } 1\ Q), X(\forall P)Y, Y \sqsubseteq A, X(\forall Q)Z,$

$Z \sqsubseteq \neg A\}$ and the role hierarchy $\mathcal{R} = \{P \leq P, Q \leq Q, R \leq R, P \leq R, Q \leq R\}$ of example 6.4.1.

Through the application of $R_{\mathcal{ALCNR}}$-completion rules we obtain following constraint systems:

$S \rightarrow_{\text{atleast}} S \cup \{xPy_1\}$

$\rightarrow_{\forall} \quad S \cup \{xPy_1, y_1{:}Y\}$

$\rightarrow_{\sqsubseteq} \quad S \cup \{xPy_1, y_1{:}Y, y_1{:}A\}$

$\rightarrow_{\text{atleast}} S \cup \{xPy_1, y_1{:}Y, y_1{:}A, xPy_2\}$

$\rightarrow_{\forall} \quad S \cup \{xPy_1, y_1{:}Y, y_1{:}A, xPy_2, y_2{:}Y\}$

$\rightarrow_{\sqsubseteq} \quad S \cup \{xPy_1, y_1{:}Y, y_1{:}A, xPy_2, y_2{:}Y, y_2{:}A\}$

$\rightarrow_{\text{atleast}} S \cup \{xPy_1, y_1{:}Y, y_1{:}A, xPy_2, y_2{:}Y, y_2{:}A, xQz\}$

$\rightarrow_{\forall} \quad S \cup \{xPy_1, y_1{:}Y, y_1{:}A, xPy_2, y_2{:}Y, y_2{:}A, xQz, z{:}Z\}$

$\rightarrow_{\sqsubseteq} \quad S \cup \{xPy_1, y_1{:}Y, y_1{:}A, xPy_2, y_2{:}Y, y_2{:}A, xQz, z{:}Z, z{:}\neg A\} = S'$

$\rightarrow_{P \leq R} \quad S' \cup \{xRy_1\}$

$\rightarrow_{P \leq R} \quad S' \cup \{xRy_1, xRy_2\}$

$\rightarrow_{P \leq R} \quad S' \cup \{xRy_1, xRy_2, xRz\} = S''$

Because $\{x{:}X, X(\text{atmost } 2\ Q)\} \subseteq S''$, we have to consider the following substitutions:

i) $S'' \rightarrow_{\text{atmost}} \{y_1 \leftarrow z\}S''$

ii) $S'' \rightarrow_{\text{atmost}} \{y_2 \leftarrow z\}S''$

iii) $S'' \rightarrow_{\text{atmost}} \{z \leftarrow y_1\}S''$

iv) $S'' \rightarrow_{\text{atmost}} \{z \leftarrow y_2\}S''$

Each of these four constraint systems contains the constraints x:A and x:¬A for some $x \in V^I$ and some concept symbol A, and we conclude that S is not $\mathcal{R}$-consistent. On the other hand, if we use the $\rightarrow_{\text{atleast}}$ - rule, which adds the single constraint xRy, then we

obtain the constraint system

S $\cup$ {xPy, y:Y, y:A, xQz, z:Z, z:$\neg$A, xRy, xRz},

that doesn't contain constraints of the form x:A and x:$\neg$A. We might come to the incorrect conclusion that S is $\mathcal{R}$-consistent.

As mentioned above, Nebel shows that $\mathcal{FL}^-$ extended simultaneously by number restrictions and the androle operator is NP-hard in the strong sense, that is, it is independent in which way the numbers are coded. The complexity of an algorithm for checking the consistency of $\mathcal{ALCNR}$- concept descriptions using traces as defined in chapter 5 and 6 depends on the coding of the numbers. Consider the fresh constraint system S = {x:X, X(atleast N R)}. To complete S with the R_1-completion rules we have to add the constraints xRy_1, ..., xRy_N. If N is a big number, then the length of the string N might be about the length of the string S.

1) If we assume that N is represented in binary (as we have done), then $\log(N) \approx \|S\|$ and $N \approx e^{\|S\|}$. Thus exponential space is needed to store the N constraints simultaneously. To prevent this we have shown in chapter 4.4. that is sufficient to add the single constraint xRy.

2) On the other hand, if we assume that N is coded in unary, then $N \approx \|S\|$. Thus linear space in the size of the input constraint system is needed to store the N constraints. As a consequence, we obtain an PSPACE-algorithm for checking consistency of $\mathcal{ALCNR}$ - concept descriptions, if the numbers are coded in unary.

7 Attributive concept descriptions and features

The restricted version of the language $\mathcal{ALC}$ obtained by interpreting all role symbols as partial functions (so-called *features*), is a subset of the Feature Logic investigated in [Smolka 88]. Smolka includes the *selection*, *agreement* and *disagreement* operators for features. The selection operator f:C takes a feature symbol f and a concept description C as arguments and denotes the set of all elements of the domain for which the feature f is defined and the application of f yields an element of the set denoted by C. Thus the selection operator corresponds to the $\forall$R:C and $\exists$R:C operator for roles. The agreement and disagreement operators are similar to KL-ONE's role value map for features as role symbols. In the following we combine the language $\mathcal{ALC}$ with Feature Logic (without subsort). Thus, we obtain an ACD-language that admits roles, which are interpreted as any relations, and features, which are interpreted as partial functions.

7.1. Syntax and semantics of $\mathcal{ALCF}$-concept descriptions

If D is a set and F: D $\rightarrow$ D is a partial function on D, then

dom F = {d $\in$ D | F is defined at d}

denotes the **domain** of F. If G: D $\rightarrow$ D is another partial function, then the composition F$\circ$G has the domain

dom F$\circ$G = {d $\in$ dom G | G(d) $\in$ dom F}.

We assume a further alphabet of symbols, called **features**, disjoint from concept and role symbols, respectively. The letters f, g will always denote a feature symbol. A **path**, denoted by p, q, is a sequence $f_1...f_n$ of feature symbols. The **empty path** is denoted by ε.

The concept descriptions of the language $\mathcal{ALCF}$ ($\mathcal{ALC}$ + $\mathcal{F}$eatures) are given by the abstract syntax rule

$$C, D \rightarrow A \mid \forall R{:}C \mid \exists R{:}C \mid C \sqcup D \mid C \sqcap D \mid \neg C \mid f{:}C \mid p{\downarrow}q \mid p{\uparrow}q\,.$$

An **interpretation** $I = (\mathcal{D}^I, I[\bullet])$ interprets every feature symbol and every path as a partial function from $\mathcal{D}^I$ to $\mathcal{D}^I$, maps every concept description to a subset of $\mathcal{D}^I$ and every role symbol to a subset of $\mathcal{D}^I \times \mathcal{D}^I$, satisfying the following equations:

- $I[\varepsilon](a) = a$ for every $a \in \mathcal{D}^I$ ($I[\varepsilon]$ is the identity function on $\mathcal{D}^I$)
- $I[fp](a) = I[p](I[f](a))$
- $I[\top] = \mathcal{D}^I$
- $I[\bot] = \emptyset$
- $I[f{:}C] = \{ a \in \mathrm{dom}\ I[f] \mid I[f](a) \in I[C] \}$
- $I[p{\downarrow}q] = \{ a \in \mathrm{dom}\ I[p] \cap \mathrm{dom}\ I[q] \mid I[p](a) = I[q](a) \}$
- $I[p{\uparrow}q] = \{ a \in \mathrm{dom}\ I[p] \cap \mathrm{dom}\ I[q] \mid I[p](a) \neq I[q](a) \}$

and the equations for $I[\forall R{:}C]$, $I[\exists R{:}C]$, $I[C \sqcup D]$, $I[C \sqcap D]$, $I[\neg C]$ as in chapter 2.

An interpretation I is a **model** for an $\mathcal{ALCF}$ - concept description (concept description, for short) C if $I[C]$ is nonempty. Furthermore, **consistency, subsumption**, and **equivalence** are defined as always.

The selection operator f:C denotes the set of all elements of the domain for which the feature f is defined and for which the application of f yields an element of the set denoted by C. The agreement operator $p{\downarrow}q$ (disagreement operator $p{\uparrow}q$) taking two paths as arguments, denotes the set of all elements of the domain for which p and q are both defined and the application of p and q yields (doesn't yield) the same element as result. Note that

$$I[p{\uparrow}q] = I[p{:}\top \sqcap q{:}\top \sqcap \neg(p{\downarrow}q)]$$

for every interpretation I.

Let R be a role symbol such that $(a,b) \in \mathcal{I}[R]$ and $(a,c) \in \mathcal{I}[R]$ implies $b = c$, that is, R denotes a partial function. Then we have

- $\mathcal{I}[\forall R{:}C] = \mathcal{I}[R{:}C \sqcup \neg(R{:}\top)]$
- $\mathcal{I}[\exists R{:}C] = \mathcal{I}[R{:}C]$.

Thus the $\forall R{:}C$ and the $\exists R{:}C$ operators can be expressed in terms of the selection operator, if R is a partial function. Conversely,

- $\mathcal{I}[f{:}C] = \mathcal{I}[\exists f{:}C]$

and the selection operator can be expressed with the $\exists R{:}C$ operator. The agreement and disagreement operators are new and enhance the expressiveness of the language.

7.2. Simplification and unfolding of $\mathcal{ALCF}$-concept descriptions

A complement is called **simple** if it has either the form $\neg A$, where A is a concept symbol different from $\top$ and $\bot$, or the form $\neg f{:}\top$.

We supplement the **simplification rules** of chapter 2 by following rules:

- $\neg f{:}C \rightarrow \neg f{:}\top \sqcup f{:}\neg C$
- $\neg p{\downarrow}q \rightarrow \neg p{:}\top \sqcup \neg q{:}\top \sqcup p{\uparrow}q$
- $\neg p{\uparrow}q \rightarrow \neg p{:}\top \sqcup \neg q{:}\top \sqcup p{\downarrow}q$.

A concept description is called **simple** if it contains only simple complements.

Proposition 7.2.1. *For every $\mathcal{ALCF}$ - concept description one can compute in linear time an equivalent simple $\mathcal{ALCF}$ - concept description.*

Proof. A simple $\mathcal{ALCF}$ - concept description can be obtained from an $\mathcal{ALCF}$ - concept description in linear time by rewriting with the simplification rules in top-down order.

Note that the simplification rules preserve consistency and inconsistency. □

As in chapter 2 we will transform $\mathcal{ALCF}$ - concept descriptions into constraint systems. We have to define further constraints for coding the selection, agreement and disagreement operators, respectively. A **constraint** has one of the following forms:

1) $X \sqsubseteq C$, $X(\forall R)Y$, $X(\exists R)Y$, $X \sqsubseteq Y \sqcup Z$, $x:X$, $x:A$, xRy

2) $X(p\downarrow q)$, $X(p\uparrow q)$, $X(f)Y$, $X(\neg f:\top)$, xpy, $x \neq y$.

The interpretation functions for constraints of the first group are as in chapter 2. For the other constraints we define the interpretation functions as follows:

- $\mathcal{I}[X(p\downarrow q)] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X) :$
  $a \in \text{dom } \mathcal{I}[p] \cap \text{dom } \mathcal{I}[q],\ \mathcal{I}[p](a) = \mathcal{I}[q](a) \}$
- $\mathcal{I}[X(p\uparrow q)] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X) :$
  $a \in \text{dom } \mathcal{I}[p] \cap \text{dom } \mathcal{I}[q],\ \mathcal{I}[p](a) \neq \mathcal{I}[q](a) \}$
- $\mathcal{I}[X(f)Y] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X) : a \in \text{dom } \mathcal{I}[f],\ \mathcal{I}[f](a) \in \alpha(Y) \}$
- $\mathcal{I}[X(\neg f:\top)] = \{ \alpha \in ASS^{\mathcal{I}} \mid \forall a \in \alpha(X) : a \notin \text{dom } \mathcal{I}[f] \}$
- $\mathcal{I}[xpy] = \{ \alpha \in ASS^{\mathcal{I}} \mid \alpha(x) \in \text{dom } \mathcal{I}[p],\ \mathcal{I}[p](\alpha(x)) = \alpha(y) \}$
- $\mathcal{I}[x \neq y] = \{ \alpha \in ASS^{\mathcal{I}} \mid \alpha(x) \neq \alpha(y) \}$.

We define **constraint systems, models, consistency, standard interpretations** and **standard $\mathcal{I}$-assignments** as usual.

Proposition 7.2.2. *Let x be an individual variable and let X be a concept variable. A simple $\mathcal{ALCF}$ - concept description is consistent if and only if the constraint system $\{x:X, X \sqsubseteq C\}$ is consistent.*

Proof. Analogous to the proof of proposition 3.1. □

A constraint system S is **simple** if for every constraint $X \sqsubseteq C$ in S the concept description C is either a concept symbol different from $\top$ and $\bot$, or a complemented concept symbol. The **unfolding rules** of chapter 3 supplemented by

- $X \sqsubseteq p{\downarrow}q \rightarrow X(p{\downarrow}q)$
- $X \sqsubseteq p{\uparrow}q \rightarrow X(p{\uparrow}q)$
- $X \sqsubseteq f{:}C \rightarrow X(f)Y, Y \sqsubseteq C,$ where Y is a new concept variable
- $X \sqsubseteq \neg f{:}\top \rightarrow X(\neg f{:}\top)$

can be used to simplify general constraint systems obtained from $\mathcal{ALCF}$ - concept descriptions to simple constraint systems.

Proposition 7.2.3. *For every constraint system S one can compute in linear time a simple constraint system S′ such that S consistent if and only if S′ is consistent.*

Proof. A simple constraint system can be obtained from a constraint system in linear time by rewriting with the unfolding rules in top-down order. Note that the unfolding rules preserve consistency and inconsisteny. □

Theorem 7.2.4. *For every $\mathcal{ALCF}$ - concept description C one can compute in linear time a fresh constraint system such that C is consistent if and only if S is consistent.*

Proof. Using the simplification and unfolding rules. □

For the construction of constraint trees for constraint systems obtained from $\mathcal{ALCF}$ - concept descriptions we adopt the following conventions:

- the constraint X(f)Y defines a "feature" edge from node X to node Y, and
- the constraints $X(p{\downarrow}q)$, $X(p{\uparrow}q)$ and $X(\neg f{:}\top)$ define $(p{\downarrow}q)$, $(p{\uparrow}q)$ and $X(\neg f{:}\top)$ as label of node X.

7.3. Propagation

Now we are going to define completion rules to complete constraint systems obtained from $\mathcal{ALCF}$ - concept descriptions.

Proposition 7.3.1. *Let S be a constraint system. Then:*

1. *if x:X, X(f)Y are in S and y is a new variable not occurring in S, then S is consistent if and only if* $S \cup \{xfy, y{:}Y\}$ *is consistent*
2. *if x:X,* $X(p{\downarrow}q)$ *are in S and y is a new variable not occurring in S, then S is consistent if and only if* $S \cup \{xpy, xqy\}$ *is consistent*
3. *if x:X,* $X(p{\uparrow}q)$ *are in S and y, z are new variables not occurring in S, then S is consistent if and only if* $S \cup \{xpy, xqz, y{\neq}z\}$ *is consistent*
4. *if xfpy is in S,* $p \neq \varepsilon$ *and z is a new variable not occurring in S, then S is consistent if and only if* $S \cup \{xfz, zpy\}$ *is consistent*
5. *if xfy, xfz are in S, then S is consistent if and only if* $\{z \leftarrow y\}S$ *is consistent*
6. *if x:X, X(¬f:T), xfy are in S, then S is inconsistent*
7. *if* $x{\neq}x$ *is in S, then S is inconsistent .*

Proof. We show the third part of the proposition. The proofs of the other parts are similar. Let $S = \{x{:}X, X(p{\uparrow}q)\} \cup S_{rest}$.

Suppose S is consistent. Then there exists an $\mathcal{I}$-assignment α with $\alpha(x) \in \alpha(X)$, for every $a \in \alpha(X)$ we have $a \in \text{dom } \mathcal{I}[p] \cap \text{dom } \mathcal{I}[q]$, and $\mathcal{I}[p](a) \neq \mathcal{I}[q](a)$. It follows that there exist $b, c \in \mathcal{D}^{\mathcal{I}}$ such that $\mathcal{I}[p](a) = b$, $\mathcal{I}[q](a) = c$ and $b \neq c$. Let y and z new individual variables. Put $\alpha'(x) = \alpha(x)$ for $x \in V^I$, $x \neq y$, $x \neq z$, $\alpha'(y) = b$, $\alpha'(z) = c$ and $\alpha'(X) = \alpha(X)$ for $X \in V^C$. Hence $\mathcal{I}[p](\alpha'(x)) = \alpha'(y)$, $\mathcal{I}[q](\alpha'(x)) = \alpha'(z)$ and $\alpha'(y) \neq \alpha'(z)$. Thus $\alpha' \in S' = S \cup \{xpy, xqz, y{\neq}z\}$ and S' is consistent.

If S is inconsistent, then $\mathcal{I}[S] = \emptyset$ and $\mathcal{I}[S'] = \mathcal{I}[S] \cap \mathcal{I}[xpy, xqz, y{\neq}z] = \emptyset$. Thus S' is not consistent. □

We define the $\mathbf{R}_{\mathcal{ALCF}}$**-completion rules** that consist of the $\rightarrow_{\forall}$ - rule, the $\rightarrow_{\exists}$ - rule, the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\sqsubseteq}$ - rule and the following rules:

$S \rightarrow_{\exists\text{-feature}} \{y:Y, xfy\} \cup S$

if x:X, X(f)Y are in S and there exists no individual variable z such that xfz is in S, and y is an individual variable not occurring in S

$S \rightarrow_{\forall\text{-feature}} \{y:Y\} \cup S$

if x:X, X(f)Y, xfy are in S and y:Y is not in S

$S \rightarrow_{\downarrow} \{xpy, xqy\} \cup S$

if x:X, X(p$\downarrow$q) are in S and there exists no individual variable z such that xpz and xqz are in S, and y is an individual variable not occurring in S

$S \rightarrow_{\uparrow} \{xpy, xqz, y \neq z\} \cup S$

if x:X, X(p$\uparrow$q) are in S and there exist no individual variables u, v such that xpu, xqv, u$\neq$v are in S, and y, z are individual variables not occurring in S

$S \rightarrow_{\text{path}} \{xfz, zpy\} \cup S$

if xfpy is in S, $p \neq \varepsilon$ and there exist no individual variables u, v such that xfu, upv are in S, and z is an individual variable not occurring in S

$S \rightarrow_{\text{function}} \{y \leftarrow z\}S$

if xfy and xfz are in S

$S \rightarrow_{\text{error}} \{x:A, x:\neg A\}$

if x:X, X($\neg$f:T), xfy are in S or if x$\neq$x is in S.

A constraint system is $\mathbf{R}_{\mathcal{ALCF}}$**-complete** if no $R_{\mathcal{ALCF}}$-completion rule applies to it.

Proposition 7.3.2. *If the constraint system S´ is obtained from the constraint system S by application of the* $\rightarrow_{\exists\text{-feature}}$ *- rule, the* $\rightarrow_{\forall\text{-feature}}$ *- rule, the* $\rightarrow_{\downarrow}$ *- rule, the* $\rightarrow_{\uparrow}$ *- rule, the* $\rightarrow_{path}$ *- rule, the* $\rightarrow_{function}$ *- rule or the* $\rightarrow_{error}$ *- rule, then S is consistent if and only if S´ is consistent.*

Proof. Follows from proposition 7.2.1. □

Proposition 7.3.3. *An $R_{\mathcal{ALCF}}$-complete constraint system is inconsistent if and only if it contains for some $x \in V^I$ and some concept symbol A, the constraints x:A and x:¬A.*

Proof. "⇒": Let S be an $R_{\mathcal{ALCF}}$-complete constraint system not containing the constraints x:A and x:¬A for any individual variable x and any concept symbol A. We show that the standard interpretation I_s of S is a model for S. In particular we show that for the standard I-assignment α_s we have $\alpha_s \in I_s[c]$ for every $c \in S$.

Suppose, $\alpha_s \notin I_s[c]$ for some $c \in S$, then the completion rule for c applies. If c has the form X(∀R)Y, X(∃R)Y, $X \sqsubseteq Y \sqcup Z$, x:X, x:A, $X \sqsubseteq A$ or xRy, then $\alpha_s \in I_s[c]$ (see proposition 4.2.1.) Furthermore $\alpha_s \in I_s[x:\neg A]$ if x:A is not in S.

Now we show that $\alpha_s \in I_s[X(p{\downarrow}q)]$. If there is no constraint x:X in S, then $\alpha_s \in I_s[X(p{\downarrow}q)]$ holds trivially. If $\{x:X, X(p{\downarrow}q)\} \subseteq S$, then the constraints xpy, xqy are in S, because S is $R_{\mathcal{ALCF}}$-complete. Hence $\alpha_s \in I_s[x:X, X(p{\downarrow}q)]$.

Next we show that $\alpha_s \in I_s[X(p{\uparrow}q)]$. If there is no constraint x:X in S, then $\alpha_s \in I_s[X(p{\uparrow}q)]$ holds trivially. If $\{x:X, X(p{\uparrow}q)\} \subseteq S$, then the constraints xpy, xqz, $y \neq z$ are in S, because S is $R_{\mathcal{ALCF}}$-complete. Hence $\alpha_s \in I_s[x:X, X(p{\uparrow}q)]$.

Furthermore we have $\alpha_s \in I_s[c]$, where c has the form X(f)Y, X(¬f:⊤), xpy or $x \neq y$.

"⇐": As in proposition 4.2.1. □

Now we will show that checking the consistency of $\mathcal{ALCF}$ - concept descriptions is decidable. To do this we prove that every fresh constraint system obtained from an $\mathcal{ALCF}$ - concept description can be extended to an $R_{\mathcal{ALCF}}$-complete constraint system preserving consistency and inconsistency. $R_{\mathcal{ALCF}}$-complete constraint systems can be checked in polynomial time for consistency (proposition 7.3.3).

Function $\mathcal{ALCF}$-**Completion** (S,i)

let S_i be the set of all constraints at level i in S

if $S_i = \emptyset$

then return S

else let $S_{\sqcup}$, $S_{\downarrow}$, $S_{\uparrow}$, S_{path}, $S_{function}$, $S_{\exists}$, $S_{\exists\text{-feature}}$, $S_{\forall\text{-feature}}$, $S_{\forall}$, $S_{\sqsubseteq}$, $S_{i\text{-complete}}$ such that

$S_i \Rightarrow_{\sqcup} S_{\sqcup} \Rightarrow_{\downarrow} S_{\downarrow} \Rightarrow_{\uparrow} S_{\uparrow} \Rightarrow_{path} S_{path} \Rightarrow_{function} S_{function} \Rightarrow_{\exists} S_{\exists} \Rightarrow_{\exists\text{-feature}}$

$S_{\exists\text{-feature}} \Rightarrow_{\forall\text{-feature}} S_{\forall\text{-feature}} \Rightarrow_{\forall} S_{\forall} \Rightarrow_{\sqsubseteq} S_{\sqsubseteq} \Rightarrow_{error} S_{i\text{-complete}}$

$\mathcal{ALCF}$ - Completion ($S \cup S_{i\text{-complete}}$, i+1)

end $\mathcal{ALCF}$ - Completion.

Proposition 7.3.4. *Suppose the call $\mathcal{ALCF}$ - Completion (S,0) returns the constraint system S', where S is a fresh constraint system obtained from an $\mathcal{ALCF}$ - concept description. Then:*

a) S' is $R_{\mathcal{ALCF}}$-complete.

b) S is consistent if S' is consistent.

c) After at most exponentially many propagation steps the function terminates.

Proof. (Sketch) a) Let S be a constraint system obtained from an $\mathcal{ALCF}$-concept description. Suppose $\mathcal{ALCF}$ - *Completion (S,i)* is called, that is, S is the argument of the i-th recursive call. Let S_i be the set of all constraints at level i. Consider the constraint system $S_{i\text{-complete}}$, which is obtained from S_i in the following manner:

$S_i \Rightarrow_{\sqcup} S_{\sqcup} \Rightarrow_{\downarrow} S_{\downarrow} \Rightarrow_{\uparrow} S_{\uparrow} \Rightarrow_{path} S_{path} \Rightarrow_{function} S_{function} \Rightarrow_{\exists} S_{\exists} \Rightarrow_{\exists\text{-feature}}$

$S_{\exists\text{-feature}} \Rightarrow_{\forall\text{-feature}} S_{\forall\text{-feature}} \Rightarrow_{\forall} S_{\forall} \Rightarrow_{\sqsubseteq} S_{\sqsubseteq} \Rightarrow_{error} S_{i\text{-complete}}$.

We will show that no $R_{\mathcal{ALCF}}$-completion rule applies to constraints at level i in $S_{i\text{-complete}}$.

Consider the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\downarrow}$ - rule, the $\rightarrow_{\uparrow}$ - rule, the $\rightarrow_{\exists}$ - rule, the $\rightarrow_{\exists\text{-feature}}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule:

Constraints of the form x:X, which have been introduced by the $\rightarrow_{\exists}$ - rule, the $\rightarrow_{\exists\text{-feature}}$ - rule, the $\rightarrow_{\forall\text{-feature}}$ - rule and the $\rightarrow_{\forall}$ - rule are at level i+1. If the constraint x:X is at level i in $S_{\text{i-complete}}$, then x:X is already in $S_{\sqcup}$. Therefore the $\rightarrow_{\sqcup}$ - rule, the $\rightarrow_{\downarrow}$ - rule, the $\rightarrow_{\uparrow}$ - rule, the $\rightarrow_{\exists}$ - rule, the $\rightarrow_{\exists\text{-feature}}$ - rule and the $\rightarrow_{\sqsubseteq}$ - rule don't apply to $S_{\text{i-complete}}$.

Consider the $\rightarrow_{\text{path}}$ - rule:

Since the $\rightarrow_{\downarrow}$ - rule and the $\rightarrow_{\uparrow}$ - rule are applied before the $\rightarrow_{\text{path}}$ - rule, the $\rightarrow_{\text{path}}$ - rule doesn't apply to $S_{\text{i-complete}}$.

Consider the $\rightarrow_{\text{function}}$ - rule:

Obviously, the $\rightarrow_{\text{function}}$ - rule doesn't apply to S_{function}. We consider the following two cases:

1) The constraints x:X, X(f)Y are in S_{function} and there is no constraint xfy for any feature f and any individual variable y in S_{function}. Then xfy, y:Y are added by the $\rightarrow_{\exists\text{-feature}}$ - rule and the $\rightarrow_{\text{function}}$ - rule doesn't apply to $S_{\exists\text{-feature}}$.

2) The constraints x:X, X(f)Y, xfy are in S_{function}. Then the $\rightarrow_{\exists\text{-feature}}$ - rule doesn't apply, but the $\rightarrow_{\forall\text{-feature}}$ - rule adding the constraint y:Y. Thus the $\rightarrow_{\text{function}}$ - rule doesn't apply to $S_{\forall\text{-feature}}$.

Furthermore, the $\rightarrow_{\forall}$ - rule, the $\rightarrow_{\sqsubseteq}$ - rule and the $\rightarrow_{\text{error}}$ - rule don't introduce constraints of the form xfy, and the $\rightarrow_{\text{function}}$ - rule doesn't apply to $S_{\text{i-complete}}$.

Consider the $\rightarrow_{\forall\text{-feature}}$ - rule and the $\rightarrow_{\forall}$ - rule:

Since the $\rightarrow_{\sqsubseteq}$ - rule and the $\rightarrow_{\text{error}}$ - rule, which are applied after the $\rightarrow_{\forall\text{-feature}}$ - rule and the $\rightarrow_{\forall}$ - rule, don't introduce constraints of the form xRy, xfy and x:X, the $\rightarrow_{\forall\text{-feature}}$ - rule and the $\rightarrow_{\forall}$ - rule don't apply to $S_{\text{i-complete}}$.

It is obvious that the $\rightarrow_{\text{error}}$ - rule doesn't apply to $S_{\text{i-complete}}$.

Thus we shown that no $R_{\mathcal{ALCF}}$-completion rule applies to constraints at level i in $S_{\text{i-complete}}$ and hence $S_{\text{i-complete}}$ is complete at level i.

Transforming a finite $\mathcal{ALCF}$-concept description into a constraint system, we obtain a finite constraint tree. Thus with induction on the level we prove that the function terminates and returns an $R_{\mathcal{ALCF}}$-complete constraint system.

b) Follows from the definition of the $R_{\mathcal{ALCF}}$-completion rules.

c) As in theorem 4.3.3. we can construct a polynomial p such that $\|S'\| \leq e^{(m+1)} * p(m)$, where m in the length of the string S. □

Theorem 7.3.5. *Let S be a fresh constraint system obtained from an $\mathcal{ALCF}$ - concept description C.*

a) If C is consistent, then there exists a computation using $\mathcal{ALCF}$-Completion, such that the call $\mathcal{ALCF}$-Completion(S,0) returns an $R_{\mathcal{ALCF}}$-complete constraint system not containing constraints of the form x:A and x:¬A.

b) If C is inconsistent, then every call $\mathcal{ALCF}$-Completion(S,0) returns an $R_{\mathcal{ALCF}}$-complete constraint system containing constraints of the form x:A and x:¬A.

Proof. Follows from the propositions 7.3.3. and 7.3.4 . □

Up to this day we couldn't find a PSPACE-algorithm for checking consistency of $\mathcal{ALCF}$ - concept descriptions. The idea behind the PSPACE-algorithms of chapter 5 and chapter 6 is, that a complete constraint system is the union of so-called traces. Traces have the property that they could be inspected independently. The following example shows that this approach is not viable in the presence of agreements.

Example 7.3.1.

Figure 7.3.1. shows a constraint tree for the constraint system

$S = \{x{:}X, X(f{\downarrow}g), X(f)Y_1, Y_1(\exists R)Z_1, Z_1 \sqsubseteq A, X(g)Y_2, Y_2(\forall R)Z_2, Z_2 \sqsubseteq \neg A\}$.

Using the $R_{\mathcal{ALCF}}$-completion rules we obtain the following constraint systems:

$S \rightarrow_{p{\downarrow}q} \{xfy, xgy\} \cup S$

$\rightarrow_{\forall\text{-feature}} \{xfy, xgy, y{:}Y_1\} \cup S$

$\rightarrow_{\forall\text{-feature}} \{xfy, xgy, y{:}Y_1, y{:}Y_2\} \cup S$

$\rightarrow_{\exists} \{xfy, xgy, y{:}Y_1, y{:}Y_2, yRz, z{:}Z_1\} \cup S$

$\rightarrow_{\sqsubseteq}$ $\{xfy, xgy, y{:}Y_1, y{:}Y_2, yRz, z{:}Z_1, z{:}A\} \cup S$

$\rightarrow_{\forall}$ $\{xfy, xgy, y{:}Y_1, y{:}Y_2, yRz, z{:}Z_1, z{:}A, z{:}Z_2\} \cup S$

$\rightarrow_{\sqsubseteq}$ $\{xfy, xgy, y{:}Y_1, y{:}Y_2, yRz, z{:}Z_1, z{:}A, z{:}Z_2, z{:}\neg A\} \cup$

Note that the constraint system S´ is obtained from S by application completion rules. Hence the $\mathcal{ALCF}$- concept description

$C = f{\downarrow}g \sqcap f{:}(\exists R{:}A) \sqcap g{:}(\forall R{:}\neg A)$ is inconsistent.

Now consider the following two traces

$T_1 = S \cup \{xfy, y{:}Y_1, yRz, z{:}Z_1, z{:}A\}$ and

$T_2 = S \cup \{xgy, y{:}Y_2, yRz, z{:}Z_2, z{:}\neg A\}$

for S, which are not independent because of the agreement $f{\downarrow}g$.

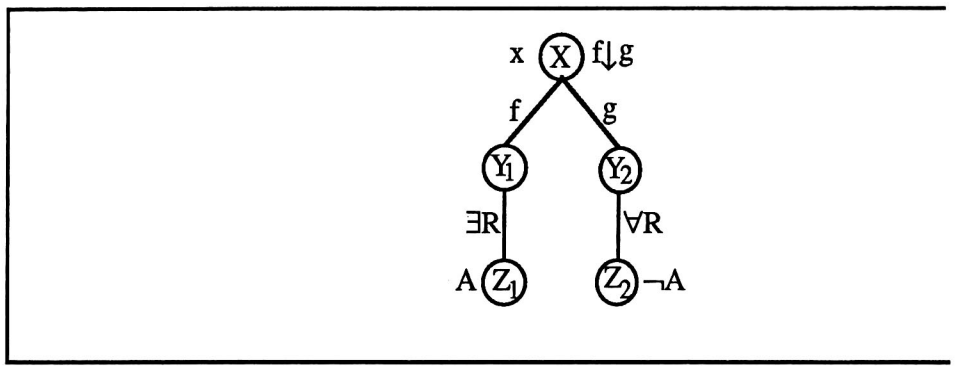

Figure 7.3.1. A constraint tree for a constraint system obtained fro concept description $C = f{\downarrow}g \sqcap f{:}(\exists R{:}A) \sqcap g{:}(\forall R{:}\neg A)$.

8 Conclusions

Many articles have been published in the last years that handle the theoretic backgrounds of knowledge representation languages based on KL-ONE, for example [Levesque Brachman 87, Nebel 88, Donini/Lenzerini 88, Schmidt-Schauß/Smolka 88]. To do reasoning in these languages it is unavoidable to have subsumption algorithms for these languages. The question, whether the problem of checking subsumption is decidable, and if so, what computational complexity it has, is very important.

One of the early results is published in [Levesque, Brachman 87]. There the language $\mathcal{FL}^-$, which is the language $\mathcal{AL}$ without simple complements, is defined and it is shown that determining subsumption in $\mathcal{FL}^-$ can be done in polynomial time. Furthermore, Levesque and Brachman define the restrict-operator with the following semantics:

I[Restrict R C] = { (a,b) ∈ $\mathcal{D}^I \times \mathcal{D}^I$ | (a,b) ∈ I[R] and b ∈ I[C] }.

It is shown that the subsumption test for $\mathcal{FL}$, which is the language $\mathcal{FL}^-$ enhanced by the restrict-operator, is NP-hard.

In his paper on feature logic, Smolka [88] shows that feature descriptions as used in computational linguistics are closely related to KL-ONE. The main difference between feature descriptions and KL-ONE concept descriptions is that in feature logic roles are interpreted as partial functions (called features) while in KL-ONE roles are interpreted as any binary relations. Notationally very similar, one minor difference in the semantics causes major computational differences. If agreement is used with roles, it causes undecidability [Schmidt-Schauß 88], while its use with features neither destroys decidability nor causes a complexity jump.

In [Schmidt-Schauß/Smolka 88] we can find some important results. In addition to complexity results on some ACD-languages they give algorithms for checking

subsumption and consistency in these languages. The algorithms work as follows: Concept descriptions that are checked for consistency are transformed into constraint systems, which are extended by completion rules. Completed constraint systems can be checked easily for consistency. Based on that idea we amalgamated the language $\mathcal{ALC}$ by number restrictions (chapter 2 - 5), role hierarchies (chapter 6), and feature logic (chapter 7), and gave consistency checking algorithms for these languages.

We proved that checking consistency of $\mathcal{ALCN}$ ($\mathcal{ALC}$ + $\mathcal{N}$umber restriction) and $\mathcal{ALCR}$ ($\mathcal{ALC}$ + $\mathcal{R}$ole hierarchy) concept descriptions, respectively, is in PSPACE. That is, the addition either of number restrictions or of role hierarchies doesn't raise the computational complexity of $\mathcal{ALC}$. On the other hand we could not find a PSPACE algorithm for checking consistency of $\mathcal{ALCNR}$ ($\mathcal{ALC}$ + $\mathcal{N}$umber restriction + $\mathcal{R}$ole hierarchy) concept descriptions (we conjecture that this problem in not in PSPACE). A similar phenomenon has been established by Nebel [88]. Checking subsumption in $\mathcal{FL}^-$ extended either by number restrictions or by the androle operator (an operator to create new roles by conjoining them - comparable to role hierarchies) can be done in polynomial time, whereas $\mathcal{FL}^-$ extended simultaneously by number restrictions and the androle operator is NP-hard.

9 References

K. H. Bläsius, H. J. Bürckert (Hrsg.): Deduktionssysteme. Oldenbourg Verlag 1987.

R. Brachman, J. Schmolze: An overview of the KL-ONE knowledge representation system. Cognitive Science 9(2) 1985, 171 - 216.

F. Donini, M. Lenzerini: TermLog : a Logic for Terminological Knowledge. 1988.

J. Hopcroft, J. Ullman: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley 1979.

H. Levesque, R. Brachman: Expressiveness and tractability in knowledge representation and reasoning. Computational Intelligence 3, 1987, 78 - 93.

K. von Luck, B. Nebel, C. Peltason, A. Schmiedel: The Anatomy of the BACK System, KIT Report 41, FB Informatik, TU Berlin, Berlin, West Germany, 1987

B. Nebel: Computational Complexity of Terminological Reasoning in BACK. Artificial Intelligence 34, 1988, 371 - 383.

B. Nebel: Reasoning and Revision in Hybrid Representation Systems. Dissertation. 1989.

A. Nijenhuis, H.Wilf: Combinatorial Algorithms. Academic Press 1975.

M. Schmidt-Schauß, G. Smolka: Attributive Concept Description with Unions and Complements. SEKI Report SR-88-21, Universität Kaiserslautern, West Germany.

M. Schmidt-Schauß: Subsumption in KL-ONE is undecidable. SEKI Report SR-88-14, Universität Kaiserslautern, West Germany. Also: Proc. of Knowledge Representation '89, Toronto, Ontario, Canada, pp. 421-431, 1989.

G. Smolka: A Feature Logic with Subsorts. LILOG Report 33, IBM Deutschland, West Germany, May 1988.